2

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-540
June 1989

Sequential Searching in Multidimensional Monotone Arrays

Alok Aggarwal and James K. Park

Abstract

Shared-memory multiprocessors commonly use shared variables for synchronization. Our simulations of real parallel applications show that large-scale cache-coherent multiprocessors suffer significant amounts of *invalidation traffic due to synchronization*. Large multiprocessors that do not cache synchronization variables are often more severely impacted. If this synchronization traffic is not reduced or managed adequately, synchronization references can cause sever congestion in the network. We propose a class of adaptive backoff methods that do not use any extra hardware and can significantly reduce the memory traffic to synchronization variables. These methods use synchronization state to reduce polling of synchronization variables. Our simulations show that when the number of processors participating in a barrier synchronization is small compared to the time of arrival of the processors, reductions of 20 percent to over 95 percent in synchronization traffic can be achieved at no extra cost. In other situations adaptive backoff techniques result in a tradeoff between reduced network accesses and increased processor idle time.

Microsystems Research Center Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

Acknowledgements

Portions of this paper appeared in the *Proceeding of the 29th Annual Symposium on Foundations of Computer Science*, 1988. This work was supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825, and the Office of Naval Research under contract N00014-86-K-0593, and an NSF Graduate Fellowship.

Author Information

Aggarwal: IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598.
Park: Laboratory for Computer Science, MIT, Room NE43-340, Cambridge, MA 02139. (617) 253-2322.

Sequential Searching in Multidimensional Monotone Arrays*

(preliminary version)

Alok Aggarwal[†] James K. Park[‡]

May 9, 1989

Abstract

A two-dimensional array is called *monotone* if the maximum entry in its i-th row lies directly below or to the right of the maximum entry in its $(i-1)$-st row. (If a row has several maxima, then we take the leftmost one.) A two-dimensional array is called *totally monotone* if every 2×2 subarray (*i.e.*, every 2×2 minor) is monotone. Totally monotone arrays were introduced by Aggarwal, Klawe, Moran, Shor, and Wilber, who showed that several problems in computational geometry could be reduced to the problem of finding the maximum entry in each row of a totally monotone array. They also gave a sequential algorithm for computing the row maxima of an $n \times n$ totally monotone array in $\Theta(n)$ time. In this paper, we generalize the notion of two-dimensional totally monotone arrays to multidimensional arrays, present sequential algorithms for finding maxima in such arrays, and exhibit a wide variety of problems (involving computational geometry, dynamic programming, and VLSI river routing) that can be solved efficiently using these array-searching algorithms.

1 Introduction

1.1 Motivation

A two-dimensional array $A = \{a_{i,j}\}$ is called *monotone* if the maximum entry in its i-th row lies directly below or to the right of the maximum entry in its $(i-1)$-st row. (If a row has several maxima, then we take the leftmost one.) A is called ***totally monotone*** if every 2×2 subarray (*i.e.*, every 2×2 minor) is monotone. In other words, for all $i < k$ and $j < l$,

*Portions of this paper appear in the Proceedings of the 29th Annual Symposium on Foundations of Computer Science, 1988.

†IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598. This work was done while the author was visiting MIT during the spring of 1988.

‡Laboratory for Computer Science, Massachusetts of Institute of Technology, Cambridge, MA 02139. Supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-825, the Office of Naval Research under Contract N00014-86-K-0593, and an NSF Graduate Fellowship.

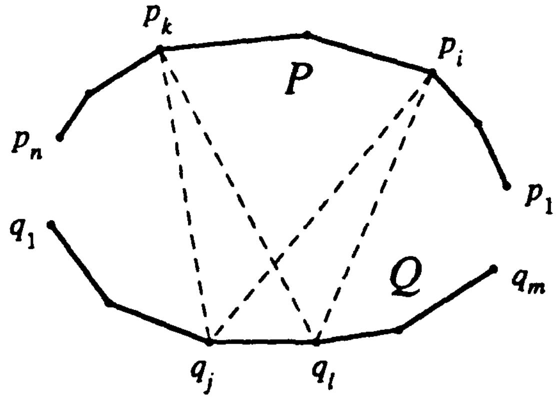

Figure 1.1: For $i < k$ and $j < l$, $d(p_i, q_j) + d(p_k, q_l) \geq d(p_i, q_l) + d(p_k, q_j)$.

$a_{i,j} < a_{i,l}$ implies $a_{k,j} < a_{k,l}$. The *row maxima* problem for A is that of finding the maximum entry in each row of the array. Now suppose some oracle has an $n \times m$ totally monotone array A and that we wish to solve the row maxima problem for A by asking the oracle for as few entries of A as possible. Aggarwal et al. [AKM*87] showed that $O(n+m)$ queries to the oracle suffice.

Although the problem of finding the row maxima in a two-dimensional totally monotone array may seem rather odd at first glance, [AKM*87] showed that several problems in computational geometry can be reduced to one or more instances of this problem. The following example illustrates one such reduction (borrowed from [AKM*87]).

Suppose we are given a convex polygon and that we divide it into two convex chains P and Q (containing n and m vertices, respectively) by removing two edges, as is shown in Figure 1.1. Let $p_1, \ldots, p_n$ denote the vertices of P in counterclockwise order and let $q_1, \ldots, q_m$ denote the vertices of Q in counterclockwise order. Then for $1 \leq i < k \leq n$ and $1 \leq j < l \leq m$, consider the quadrilateral formed by p_i, p_k, q_j, and q_l. By the ***quadrangle inequality*** (which states that the sum of the lengths of the diagonals of any quadrilateral is greater than the sum of the lengths of any pair of opposite sides), we have

$$d(p_i, q_j) + d(p_k, q_l) \;\geq\; d(p_i, q_l) + d(p_k, q_j).$$

Thus, if we imagine an $n \times m$ array $A = \{a_{i,j}\}$ where $a_{i,j}$ is the Euclidean distance from vertex $p_i \in P$ to vertex $q_j \in Q$, then this array is totally monotone, since we cannot have both $a_{i,j} < a_{i,l}$ and $a_{k,j} \geq a_{k,l}$. Also, since any entry of this array can be computed in constant time (it being the Euclidean distance between two points), asking for an entry from an oracle simply corresponds to evaluating the distance between the corresponding points. Hence, by using [AKM*87], we can find the farthest vertex in Q for every vertex in P by evaluating only $O(n+m)$ distances. *In fact, [AKM*87] showed that the time required in addition to that for evaluating these $O(n+m)$ distances is also linear in $n+m$.* This implies that the farthest neighbor problem for convex chains can be solved in linear time and solves an open problem in [Tou83].

1.2 Previous Results

As the previous example illustrates, the notion of total monotonicity is closely related to the quadrangle inequality. (It is this relation that makes the combinatorial formulation of problems in terms of totally monotone arrays such a useful one.) Another related property of two-dimensional arrays, even more closely tied to the quadrangle inequality, was studied previously in the context of certain transportation problems. In the late eighteenth century, G. Monge [Mon81] observed that if unit quantities (cannonballs, for example) need to be transported from locations X and Y (supply depots) in the plane to locations Z and W (artillery batteries), not necessarily respectively, in such a way as to minimize the total distance traveled, then the paths followed in transporting these quantities must not intersect. In 1961, A.J. Hoffman [Hof61] elaborated upon this idea by calling an $n \times m$ array $C = \{c_{i,j}\}$ a *Monge* array if $c_{i,j} + c_{i+1,j+1} \le c_{i,j+1} + c_{i+1,j}$ for $1 \le i < n$ and $1 \le j < m$ and by providing a greedy algorithm for the transportation problem when the cost array $C = \{c_{i,j}\}$ is a Monge array[1]. Monge and Hoffman's observations together imply a greedy algorithm for the transportation problem when the sources lie on one line in the plane and the sinks lie on a second, parallel line.

Note that an equivalent way to define an $n \times m$ Monge array $C = \{c_{i,j}\}$ is to require $c_{i,j} + c_{k,l} \le c_{i,l} + c_{k,j}$ for $1 \le i < k \le n$ and $1 \le j < l \le m$, as the following proposition shows.

Proposition 1.1 *If $C = \{c_{i,j}\}$ is an $n \times m$ Monge array, then for $1 \le i < k \le n$ and $1 \le j < l \le m$, $c_{i,j} + c_{k,l} \le c_{i,l} + c_{k,j}$.*

Proof Consider any i, j, k, and l such that $1 \le i < k \le n$ and $1 \le j < l \le m$. By the definition of a Monge array,

$$c_{s,t} + c_{s+1,t+1} \le c_{s,t+1} + c_{s+1,t}$$

for $1 \le s < n$ and $1 \le t < m$. Thus, for $1 \le t < m$,

$$\sum_{s=i}^{k-1}(c_{s,t} + c_{s+1,t+1}) \le \sum_{s=i}^{k-1}(c_{s,t+1} + c_{s+1,t}).$$

Cancelling identical terms from both sides of this inequality, we obtain

$$c_{i,t} + c_{k,t+1} \le c_{i,t+1} + c_{k,t}.$$

Consequently,

$$\sum_{t=j}^{l-1}(c_{i,t} + c_{k,t+1}) \le \sum_{t=j}^{l-1}(c_{i,t+1} + c_{k,t}).$$

Again cancelling identical terms, we obtain

$$c_{i,j} + c_{k,l} \le c_{i,l} + c_{k,j},$$

[1]In fact, Hoffman [Hof61] provided a necessary and sufficient condition that the cost array C must satisfy for his greedy algorithm to work. Furthermore, Alon, Cosares, Hochbaum, and Shamir [ACHS88] have recently developed an efficient algorithm for testing whether this necessary and sufficient condition holds.

the desired result. ■

We will follow Hoffman's terminology and call $c_{i,j} + c_{k,l} \leq c_{i,l} + c_{k,j}$ the *Monge condition* and $c_{i,j} + c_{k,l} \geq c_{i,l} + c_{k,j}$ the *inverse Monge condition.* Note that the inverse Monge condition implies total monotonicity, but not vice-versa. Although total monotonicity is a somewhat weaker constraint than the inverse Monge condition, it turns out that all of the applications given in this paper, as well as those given in [AKM*87, Wil88], obey the Monge or inverse Monge condition. However, we usually require only total monotonicity to obtain our results.

We will call an array Monge if it follows either the Monge condition or the inverse Monge condition, since an array satisfying the Monge condition may be converted into an array satisfying the inverse Monge condition (and vice-versa) by reversing the order of its columns. Note that in a similar fashion, we can flip the signs in our definition of total monotonicity (at least in the case of two-dimensional arrays) and convert back and forth between totally monotone arrays and arrays $A = \{a_{i,j}\}$ such that $a_{i,j} > a_{i,l}$ implies $a_{k,j} > a_{k,l}$ for all $i < k$ and $j < l$. Furthermore, the problem of finding the maximum entry in each row of a two-dimensional array A and the problem of finding the minimum entry in each row of A (the *row minima* problem) are equivalent — we can convert one to the other by simply negating the entries of A and reversing the order of its columns. We make use of these dualities throughout this paper.

As mentioned earlier, totally monotone arrays were first introduced by Aggarwal, Klawe, Moran, Shor, and Wilber in [AKM*87], who provided a linear time algorithm for computing the row maxima in such arrays. Previous applications of this array-searching algorithm include the following. [AKM*87] showed that some channel routing problems can be solved in linear time using these techniques, thereby improving previous $O(n \lg n)$ time algorithms. Aggarwal and Suri [AS87] used array-searching to find the largest empty rectangle in $O(n \lg^2 n)$ time. Wilber [Wil88] applied array-searching techniques to an instance of dynamic programming and showed that the concave least-weight subsequence problem can be solved in linear time; this improved the result of Hirschberg and Larmore [HL87]. Finally, Klawe and Kleitman [KK88] used array-searching to improve a previous result of [AK88] in computational geometry.

1.3 Main Results of this Paper

In this paper, we present a framework that allows us to efficiently solve a wide variety of problems involving the quadrangle inequality. We limit ourselves to sequential computation; results in the realm of parallel computation are given in [AP89]. This paper derives its primary inspiration from [AKM*87] and [Yao80]; it generalizes and incorporates several results provided in these papers. (The relation of [AKM*87] to this paper will be observed throughout, but that of [Yao80] is more implicit. We assume total familiarity of the reader with [AKM*87] and [Yao80].)

The balance of this paper is organized as follows.

Section 2 introduces the notion of multidimensional monotone arrays and provides sequential algorithms for searching in such arrays. We use the problem of finding a maximum perimeter d-gon inscribed in a given convex n-gon as a prototype example. For this problem, we achieve the time bound of [AKM*87].

Section 3 applies the monotone array framework to the problem of finding minimum area and minimum perimeter circumscribing polygons. For the minimum area circumscribing d-gon problem, we obtain an $O(dn + n \lg n)$ time algorithm, thereby improving the best previous result of [AKM*87] by a factor of $O((n \lg d)/(d + \lg n))$. For the minimum perimeter circumscribing triangle problem, we obtain an $O(n \lg n)$ time algorithm, thereby improving the result of [DeP87] by an $O(n^2/\lg n)$ factor.

Section 4 presents efficient algorithms for dynamic programming problems that obey the Monge condition or the inverse Monge condition. We give an $O(n^2)$ time algorithm for Frances Yao's [Yao80] dynamic programming problem using the monotone array framework. Although our algorithm is no better than Yao's in terms of asymptotic complexity (and probably worse in practice), it provides insight into other dynamic programming problems that obey Monge conditions. We then use this insight to reduce the time complexity of a particular dynamic programming problem related to biology from $O(n^2 \lg^2 n)$ [EGG88] to $O(n^2 \lg n)$.

Section 5 presents efficient sequential algorithms for river routing in VLSI. We show that the results of [AKM*87] can be used to solve the offset range problem in linear time under very weak assumptions regarding the routing rules for wires. This generalizes a recent result of Siegel [Sie88], who was able to obtain a linear time algorithm for only a very restricted class of wiring rules.

Section 6 discusses open problems.

2 Multidimensional Monotone Arrays

2.1 Basic Definitions and Algorithms

We begin by extending our notions of monotone, totally monotone, and Monge arrays to higher dimensions. For $d \geq 3$, let $A = \{a_{i_1,i_2,\ldots,i_d}\}$ be an $n_1 \times n_2 \times \cdots \times n_d$ array. Let $i_2(i_1), \ldots, i_d(i_1)$ denote the second through d-th coordinates of the maximum entry in the $(d-1)$-dimensional plane consisting of those entries whose first coordinate is i_1, *i.e.*,

$$a_{i_1,i_2(i_1),\ldots,i_d(i_1)} = \max_{i_2,\ldots,i_d} a_{i_1,i_2,\ldots,i_d}.$$

(If the plane corresponding to i_1 contains multiple maxima, we chose the first of the maxima ordered lexicographically by their second through d-th coordinates.) We call these entries *plane maxima*.

Definition 2.1 *A is* monotone *if*

1. *for $1 \leq i_1 < j_1 \leq n_1$, we have $i_k(i_1) \leq i_k(j_1)$ for all k between 2 and d, and*

2. *for $1 \leq i_1 \leq n_1$, the $(d-1)$-dimensional plane of A consisting of those entries whose first coordinate is i_1 is monotone.*

Definition 2.2 *A is* totally monotone *if every $2 \times 2 \times \cdots \times 2$ d-dimensional subarray of A is monotone.*

Definition 2.3 *A satisfies the* Monge condition *if every two-dimensional plane of A corresponding to fixed values of $d-2$ of A's d coordinates satisfies the Monge condition. Similarly, A satisfies the* inverse Monge condition *if every two-dimensional plane of A corresponding to fixed values of $d-2$ of A's d coordinates satisfies the inverse Monge condition. Finally, A is* Monge *if it satisfies either the Monge condition or the inverse Monge condition.*

Note that every subarray of a totally monotone array is also totally monotone. Similarly, every subarray of an array satisfying the Monge condition satisfies the Monge condition, and every subarray of an array satisfying the inverse Monge condition satisfies the inverse Monge condition.

Proposition 2.4 *Suppose A satisfies the inverse Monge condition, and consider any two entries $a_{i_1,i_2,\ldots,i_d}$ and $a_{j_1,j_2,\ldots,j_d}$ of A. For $1 \le k \le d$, let $x_k = \min\{i_k, j_k\}$ and let $y_k = \max\{i_k, j_k\}$. Then*

$$a_{x_1,x_2,\ldots,x_d} + a_{y_1,y_2,\ldots,y_d} \ge a_{i_1,i_2,\ldots,i_d} + a_{j_1,j_2,\ldots,j_d}.$$

Proof We prove this proposition by induction on d. For the base case of $d = 2$, the proposition follows immediately from the definition of a two-dimensional array satisfying the inverse Monge condition.

Now suppose the proposition holds for all $(d-1)$-dimensional arrays satisfying the inverse Monge condition, and consider a d-dimensional array A satisfying the inverse Monge condition and any two entries $a_{i_1,i_2,\ldots,i_d}$ and $a_{j_1,j_2,\ldots,j_d}$ from A. Without loss of generality, we assume $i_1 < j_1$. The proof then breaks down into two cases.

Case 1 For all k between 2 and d, $i_k > j_k$ (*i.e.*, $x_k = j_k$ and $y_k = i_k$).

Consider the $(d-1)$-dimensional subarray of A containing those entries whose second coordinate is i_2 and the $(d-1)$-dimensional subarray of A containing those entries whose second coordinate is j_2. Since every subarray of an array satisfying the inverse Monge condition satisfies the inverse Monge condition, we can invoke the inductive hypothesis on these subarrays and obtain

$$a_{i_1,i_2,j_3,\ldots,j_d} + a_{j_1,i_2,i_3,\ldots,i_d} \ge a_{i_1,i_2,i_3,\ldots,i_d} + a_{j_1,i_2,j_3,\ldots,j_d}$$

and

$$a_{i_1,j_2,j_3,\ldots,j_d} + a_{j_1,j_2,i_3,\ldots,i_d} \ge a_{i_1,j_2,i_3,\ldots,i_d} + a_{j_1,j_2,j_3,\ldots,j_d}.$$

Similarly, we can invoke the inductive hypothesis on the two-dimensional subarray containing those entries whose third through d-th coordinates are $i_3, \ldots, i_d$, respectively, and on the two-dimensional subarray containing those entries whose third through d-th coordinates are $j_3, \ldots, j_d$, respectively. This gives us

$$a_{i_1,j_2,i_3,\ldots,i_d} + a_{j_1,i_2,i_3,\ldots,i_d} \ge a_{i_1,i_2,i_3,\ldots,i_d} + a_{j_1,j_2,i_3,\ldots,i_d}$$

and

$$a_{i_1,j_2,j_3,\ldots,j_d} + a_{j_1,i_2,j_3,\ldots,j_d} \ge a_{i_1,i_2,j_3,\ldots,j_d} + a_{j_1,j_2,j_3,\ldots,j_d}.$$

Summing these four inequalities and cancelling, we find

$$2a_{i_1,j_2,j_3,\ldots,j_d} + 2a_{j_1,i_2,i_3,\ldots,i_d} \;\geq\; 2a_{i_1,i_2,i_3,\ldots,i_d} + 2a_{j_1,j_2,j_3,\ldots,j_d}.$$

Since $x_1 = i_1$, $y_1 = j_2$, and for $2 \leq k \leq d$, $x_k = j_k$ and $y_k = i_k$, this gives the desired result.

Case 2 There exists an l, $2 \leq l \leq n$, such that $i_l < j_l$ (*i.e.*, $x_l = i_l$ and $y_l = j_l$).

Consider the $(d-1)$-dimensional subarray containing those entries whose first coordinate is i_1 and the $(d-1)$-dimensional subarray containing those entries whose first coordinate is j_1. By applying the inductive hypothesis to these subarrays, we obtain

$$a_{i_1,x_2,x_3,\ldots,x_d} + a_{i_1,y_2,y_3,\ldots,y_d} \;\geq\; a_{i_1,i_2,i_3,\ldots,i_d} + a_{i_1,j_2,j_3,\ldots,j_d}$$

and

$$a_{j_1,x_2,x_3,\ldots,x_d} + a_{j_1,y_2,y_3,\ldots,y_d} \;\geq\; a_{j_1,i_2,i_3,\ldots,i_d} + a_{j_1,j_2,j_3,\ldots,j_d}.$$

Similarly, by applying the inductive hypothesis to the $(d-1)$-dimensional subarray containing those entries whose l-th coordinate is i_l and to the $(d-1)$-dimensional subarray containing those entries whose l-th coordinate is j_l, we obtain

$$a_{i_1,x_2,x_3,\ldots,x_d} + a_{j_1,i_2,i_3,\ldots,i_d} \;\geq\; a_{i_1,i_2,i_3,\ldots,i_d} + a_{j_1,x_2,x_3,\ldots,x_d}$$

and

$$a_{i_1,j_2,j_3,\ldots,j_d} + a_{j_1,y_2,y_3,\ldots,y_d} \;\geq\; a_{i_1,y_2,y_3,\ldots,y_d} + a_{j_1,j_2,j_3,\ldots,j_d}.$$

Summing these four inequalities and cancelling, we find

$$2a_{i_1,x_2,x_3,\ldots,x_d} + 2a_{j_1,y_2,y_2,\ldots,y_d} \;\geq\; 2a_{i_1,i_2,i_3,\ldots,i_d} + 2a_{j_1,j_2,j_3,\ldots,j_d}.$$

Since $x_1 = i_1$ and $y_1 = j_1$, this gives the desired result. ■

Lemma 2.5 *If A satisfies the inverse Monge condition, then A is totally monotone.*

Proof We again use induction on d. For the base case of $d = 2$, the lemma follows immediately, since every two-dimensional array satisfying the inverse Monge condition is totally monotone.

Now suppose the lemma holds for all $(d-1)$-dimensional arrays, and consider a d-dimensional array A satisfying the inverse Monge condition. Since every subarray of an array satisfying the inverse Monge condition satisfies the inverse Monge condition, every $(d-1)$-dimensional plane of A consisting of those entries whose first coordinate is some fixed value satisfies the inverse Monge condition. Thus, by the induction hypothesis, each of these planes is totally monotone. This means A satisfies the second property of totally monotone arrays.

Now suppose A does not satisfy the first property of totally monotone arrays, *i.e.*, there exist i_1, j_1, and l, $1 \leq i_1 < j_1 \leq n_1$ and $2 \leq l \leq d$, such that $i_l(i_1) > i_l(j_1)$. For $2 \leq k \leq d$, let $x_k = \min\{i_k(i_1), i_k(j_1)\}$ and let $y_k = \max\{i_k(i_1), i_k(j_1)\}$. By Proposition 2.4,

$$a_{i_1,x_2,\ldots,x_d} + a_{j_1,y_2,\ldots,y_d} \;\geq\; a_{i_1,i_2(i_1),\ldots,i_d(i_1)} + a_{j_1,i_2(j_1),\ldots,i_d(j_1)}.$$

This gives us a ntradiction, as the definition of plane maxima requires

$$a_{i_1,i_2(i_1),\ldots,i_d(i_1)} > a_{i_1,x_2,\ldots,x_d}$$

and

$$a_{j_1,i_2(j_1),\ldots,i_d(j_1)} \geq a_{j_1,y_2,\ldots,y_d}.$$

■

Note that the converse of Lemma 2.5 is false, *i.e.*, totally monotone arrays do not necessarily satisfy the inverse Monge condition. Thus, total monotonicity is a weaker condition, just as it was in the two-dimensional case.

We call the problem of computing the plane maxima of an array A the *plane maxima problem* for A. The following theorem gives an upper bound on the time necessary to solve the plane maxima problem for a totally monotone array.

Theorem 2.6 *For $d \geq 2$, the plane maxima problem for an $n_1 \times n_2 \times \cdots \times n_d$ d-dimensional totally monotone array A can be solved in $O((n_d + n_{d-1})\prod_{k=1}^{d-2}\lg n_k)$ time.*

Proof The proof is by induction on d. For the base case of $d = 2$, we simply apply the $O(n)$ time algorithm of [AKM*87]. For larger values of d, we use a simple divide-and-conquer approach. We begin by computing the maximum entry in the plane corresponding to $i_1 = \lceil n_1/2 \rceil$. By induction, this can be done in $O((n_d + n_{d-1})\prod_{k=2}^{d-2}\lg n_k)$ time — we simply compute the plane maxima in this plane and then spend an additional $O(n)$ time to compute the maximum of these maxima. This gives us $i_k(i_1)$ for $2 \leq k \leq d$. Now since A is monotone, we know that for $j_1 < i_1$, we must have $i_k(j_1) \leq i_k(i_1)$ for $2 \leq k \leq d$, and similarly for $j_1 > i_1$, we must have $i_k(j_1) \geq i_k(i_1)$ for $2 \leq k \leq d$. Thus, we need only consider two smaller arrays, one $(i_1 - 1) \times i_2(i_1) \times \cdots \times i_d(i_1)$ and the other $(n_1 - i_1) \times (n_2 - i_2(i_1) + 1) \times \cdots \times (n_d - i_d(i_1) + 1)$, in computing the remaining plane maxima. This gives a recurrence for the time $T(n_1, n_2, \ldots, n_d)$ necessary to solve the plane maxima problem for A whose solution is $O((n_d + n_{d-1})\prod_{k=1}^{d-2}\lg n_k)$. ■

Note that the only place we take advantage of A's totally monotonicity is in solving the row maxima problem for certain two-dimensional subarrays — we can do almost as well (*i.e.*, compute the plane maxima of A in $O(n_d\prod_{k=2}^{d-1}\lg n_k)$ time) using only monotonicity. Also note that finding the plane minima in a totally monotone array A is no harder that finding the plane maxima — we just negate all of A's entries as we did in the two-dimensional case and reverse the order of some of its coordinates. However, we cannot convert back and forth between arrays satisfying the Monge and inverse Monge conditions, as we could in the case of two dimensional arrays.

2.2 An Example

To motivate the definitions of the previous subsection, let us consider a concrete example. The maximum perimeter inscribed d-gon problem is defined as follows: given an n-vertex convex polygon P, find a d-gon Q contained in P with maximum perimeter. It is easy to show that Q's vertices must be vertices of P. Thus, if $p_1, \ldots, p_n$ denote the vertices of P

and $\mathrm{per}(i_1, i_2, \ldots, i_d)$ denotes the perimeter of the d-gon corresponding to $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$, we want to find the $i_1, i_2, \ldots, i_d$ maximizing $\mathrm{per}(i_1, i_2, \ldots, i_d)$.

For the case of $d = 3$ (the maximum perimeter inscribed triangle problem), we consider the three-dimensional array $A = \{a_{i,j,k}\}$ where

$$a_{i,j,k} = \begin{cases} \mathrm{per}(i,j,k) & \text{if } i < j < k \\ -\infty & \text{otherwise.} \end{cases}$$

If we can find the maximum entry in A, we can solve the maximum perimeter inscribed triangle problem for P. *Note that we do not explicitly compute all of the entries in A; rather, every time our array-searching algorithm needs a particular entry from A, we just calculate the perimeter of the corresponding triangle.*

Now it is easy to verify that the array A defined above satisfies the inverse Monge condition. Furthermore, obtaining a particular entry in the array requires only constant time. Thus, we can apply Theorem 2.6 and obtain the plane maxima of A in $O(n \lg n)$ time. We can then compute the maximum of these maxima in $O(n)$ additional time, which gives a maximum perimeter inscribed triangle.

This result equals the result obtained by Boyce et al. in [BDDG85]; moreover, it represents a simpler solution to the problem, as Boyce et al. require a number of additional geometric properties of inscribed triangles that complicate their proof.

In a similar fashion, the maximum perimeter inscribed d-gon problem can be reduced to the problem of finding the maximum entry in the d-dimensional array $A = \{a_{i_1, i_2, \ldots, i_d}\}$ where

$$a_{i_1, i_2, \ldots, i_d} = \begin{cases} \mathrm{per}(i_1, i_2, \ldots, i_d) & \text{if } i_1 < i_2 < \cdots < i_d \\ -\infty & \text{otherwise.} \end{cases}$$

It is easy to verify that A satisfies the inverse Monge condition. Furthermore, $O(d)$ time suffices to compute any entry in A. Thus, we can apply Theorem 2.6 and obtain an $O(dn \lg^{d-2} n)$ time algorithm for this problem.

It remains open whether one can do better than $O(n \lg n)$ time for the inscribed triangle problem, but the general inscribed d-gon problem can be solved more efficiently if we take advantage of some additional structure of the corresponding d-dimensional array; this additional structure is discussed in the next two subsections.

2.3 Monge-Composite Arrays

An important subclass of multidimensional totally monotone arrays consists of what we call *Monge-composite* arrays. As one might expect, an array is Monge-composite if it is composed of two-dimensional Monge arrays. (These arrays may be either variety of Monge array, *i.e.*, they can obey either the Monge condition or the inverse Monge condition, but they all must be of the same variety.) More precisely, a d-dimensional Monge-composite array is the sum of d-dimensional extensions of two-dimensional Monge arrays, where we define the sum of two arrays and the extension of an array as follows.

Let $A = \{a_{i_1, \ldots, i_d}\}$ and $B = \{b_{i_1, \ldots, i_d}\}$ be $n_1 \times \cdots \times n_d$ d-dimensional arrays. The sum of A and B (written $A + B$) is the $n_1 \times \cdots \times n_d$ d-dimensional array $C = \{c_{i_1, \ldots, i_d}\}$ where

$$c_{i_1, \ldots, i_d} = a_{i_1, \ldots, i_d} + b_{i_1, \ldots, i_d},$$

for all $i_1, \ldots, i_d$.

Now let $E = \{e_{i_1,\ldots,i_d}\}$ be an $n_1 \times \cdots \times n_d$ d-dimensional array. For any dimension k between 1 and $d+1$ and any size $\hat{n}$, the

$$n_1 \times \cdots \times n_{k-1} \times \hat{n} \times n_k \times \cdots \times n_d$$

$(d+1)$-dimensional array $F = \{f_{i_1,\ldots,i_{d+1}}\}$ is an extension of E if

$$f_{i_1,\ldots,i_{k-1},i_k,i_{k+1},\ldots,i_{d+1}} = e_{i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_{d+1}},$$

for all $i_1, \ldots, i_{d+1}$. (F is just $\hat{n}$ copies of E, each one a plane of F corresponding to some fixed value of F's k-th coordinate.) Furthermore, any extension of an extension of E is also an extension of E.

From these definitions, it is clear that each entry of a d-dimensional Monge-composite array $A = \{a_{i_1,\ldots,i_d}\}$ may be written

$$a_{i_1,\ldots,i_d} = \sum_{k<l} w^{(k,l)}_{i_k,i_l},$$

where for all $k < l$, the $n_k \times n_l$ array $W^{(k,l)} = \{w^{(k,l)}_{i_k,i_l}\}$ is a Monge array.

Proposition 2.7 *The sum of two Monge arrays is also Monge.*

Proposition 2.8 *The two-dimensional extension of a vector or a scalar is Monge.*

Proposition 2.9 *Every Monge-composite array A is Monge.*

Proof We must show that all two-dimensional planes of A, corresponding to fixed values of $d-2$ of A's d coordinates, are Monge. To see why this is true, consider any such plane. This plane is the sum of a two-dimensional Monge array, some vectors, and some scalars; thus, the plane is Monge. ∎

Two special cases of Monge-composite arrays are *path- and cycle-decomposable* arrays. An array $A = \{a_{i_1,\ldots,i_d}\}$ is path-decomposable if each of its entries satisfies

$$a_{i_1,\ldots,i_d} = w^{(1,2)}_{i_1,i_2} + w^{(2,3)}_{i_2,i_3} + \cdots + w^{(d-1,d)}_{i_{d-1},i_d},$$

where the w's are entries from two-dimensional Monge arrays $W^{(k,l)}$ as before. A is cycle-decomposable if each of its entries satisfies

$$a_{i_1,\ldots,i_d} = w^{(1,2)}_{i_1,i_2} + \cdots + w^{(d-1,d)}_{i_{d-1},i_d} + w^{(d,1)}_{i_d,i_1}.$$

Theorem 2.10 *The plane maxima of an $n_1 \times \cdots \times n_d$ d-dimensional path-decomposable array A can be computed in $O(\sum_{k=1}^{d} n_k)$ time.*

Proof The proof is by induction on d. The base case of $d = 2$ follows immediately from [AKM*87]. For $d > 2$, we assume by induction that the theorem holds for all lower-dimensional path-decomposable arrays. Since $A = \{a_{i_1,\ldots,i_d}\}$ is path-decomposable, we can write

$$a_{i_1,\ldots,i_d} = w^{(1,2)}_{i_1,i_2} + w^{(2,3)}_{i_2,i_3} + \cdots + w^{(d-1,d)}_{i_{d-1},i_d},$$

where the w's are entries from two-dimensional Monge arrays. Now consider the $n_2 \times \cdots \times n_d$ $(d-1)$-dimensional array $B = \{b_{i_2,\ldots,i_d}\}$ where

$$b_{i_1,\ldots,i_d} = w^{(2,3)}_{i_2,i_3} + \cdots + w^{(d-1,d)}_{i_{d-1},i_d}.$$

By induction, we can find the plane maxima of B in $O(\sum_{k=2}^{d} n_k)$ time. Since the maximum entry in the plane of A corresponding to a particular value $\hat{\imath}_1$ of the first coordinate is just

$$\min_{i_2}\left\{w^{(1,2)}_{i_1,i_2} + \min_{i_3,\ldots,i_d}\left\{w^{(2,3)}_{i_2,i_3} + \cdots + w^{(d-1,d)}_{i_{d-1},i_d}\right\}\right\},$$

we need only find the row maxima in the sum of $W^{(1,2)}$ and the appropriate two-dimensional extension of the vector of B's plane maxima. Since this sum is a Monge array, we can find its row maxima in an additional $O(n_1 + n_2)$ time, which yields the desired result. ■

Theorem 2.11 *The plane maxima of an $n_1 \times \cdots n_d$ d-dimensional cycle-decomposable array A can be computed in $O((\sum_{k=2}^{d} n_k)\lg n_1)$ time.*

Proof First note that each plane $A_{\hat{\imath}_1}$ of A, corresponding to a fixed value $\hat{\imath}_1$ of A's first coordinate, is path-decomposable, since

$$A_{\hat{\imath}_1} = \hat{W}^{(2,3)} + W^{(3,4)} + \cdots + W^{(d-1,d)},$$

where $\hat{W}^{(2,3)}$ is the sum of $W^{(2,3)}$ and two-dimensional extensions of the vectors $V^2 = \{w^{(1,2)}_{\hat{\imath}_1,i_2}\}$ and $V^d = \{w^{(d,1)}_{i_d,\hat{\imath}_1}\}$.

This means we can compute a particular plane's maximum entry in $O(\sum_{k=2}^{d} n_k)$ time. Combining this with the divide-and-conquer approach of Theorem 2.6, we obtain the desired result. ■

2.4 More Examples

Returning to the maximum perimeter inscribed d-gon problem, it is easy to verify that the d-dimensional array A defined in Subsection 2.2 is cycle-decomposable. Thus, we can apply Theorem 2.11 and obtain an $O(dn \lg n)$ time algorithm for the problem. Furthermore, it is possible to reduce this time bound to $O(dn + n \lg n)$ using one additional technique from [BDDG85] which allows us to restrict our attention to a relatively small subarray of A. (We present this technique in Subsection 3.1 in the context of the minimum perimeter circumscribing d-gon problem.)

As Boyce et al. [BDDG85] note, the same approach can also be used to solve the maximum area inscribed d-gon problem in $O(dn + n \lg n)$ time. Although the corresponding

d-dimensional array *does not* satisfy the inverse Monge condition (nor is it totally monotone), it can be shown that we only need to consider certain subarrays of this array, and these subarrays do satisfy the inverse Monge condition.

Another application in which multidimensional Monge-composite arrays naturally arise is the following problem: given a convex n-gon P with vertices $p_1, p_2, \ldots, p_n$ in counterclockwise order, find for each vertex p_i of P the longest d-link convex path from p_1 to p_i. Corresponding to this problem is a $(d-1)$-dimensional path-decomposable array, whose plane maxima are these maximal paths. Thus, we can apply Theorem 2.10 and obtain these paths in $O(dn)$ time. (This result is implicit in [AKM*87].)

Note that not all applications involve path- or cycle-decomposable arrays; consider, for example, the problem of finding a maximum weight d-clique of vertices from a convex n-gon, where the weight of a clique is defined to be the sum over all pairs of vertices in the clique of the distance between the two vertices. An instance of this problem corresponds to a d-dimensional Monge-composite array, each of whose entries may be computed in $O(d^2)$ time, and thus may be solved in $O(d^2 n \lg^{d-2} n)$ time. Furthermore, the d-dimensional array we consider in Subsection 3.2 in connection with the minimum perimeter circumscribing triangle problem is not even Monge-composite.

3 Finding Minimum Circumscribing Polygons

In this section, we apply algorithms for searching in totally monotone arrays to the problem of finding minimum area and minimum perimeter circumscribing polygons. Given an n-vertex convex polygon P and an integer d between 3 and n, we want to find a minimum area or minimum perimeter d-gon Q containing P. Note that for both area and perimeter minimization, Q must clearly be convex, and each of its d edges must contact P. Also note that if we can find area- and perimeter-optimal d-gons circumscribing convex n-gons, then we can also find area- and perimeter-optimal convex d-gons containing arbitrary sets of points in the plane, since any convex polygon containing a set of points must contain the points' convex hull.

Chang and Yap [CY84] showed that a minimum area circumscribing d-gon can be found in $O(n^3 \lg d)$ time using dynamic programming. Aggarwal, Chang, and Yap [ACY85] improved this result to $O(n^2 \lg d \lg n)$ time, and it is further improved to $O(n^2 \lg d)$ in [AKM*87]. We extend (in a non-trivial manner) the techniques of Boyce et al. [BDDG85] for finding maximum perimeter inscribed d-gons to obtain an $O(dn + n \lg n)$ time algorithm for the minimum area circumscribing d-gon problem. We present this algorithm in Subsection 3.1.

As for the perimeter minimization problem, the only known results are for $d = 3$. DePano [DeP87] showed that the minimum perimeter circumscribing triangle can be computed in $O(n^3)$ time. In Subsection 3.2, we improve this to $O(n \lg n)$ time using a straightforward application of our techniques for searching in three-dimensional totally monotone arrays.

In describing our algorithms for finding minimum area and minimum perimeter circumscribing d-gons, we use the following conventions. For any convex m-gon R, we let $v_1^R, \ldots, v_m^R$ and $e_1^R, \ldots, e_m^R$ denote R's vertices and edges, respectively, in counterclockwise order, where e_i^R connects v_i^R and $v_{(i+1) \bmod m}^R$. We use the letter P for the convex n-gon to be circumscribed and the letter Q for convex d-gons circumscribing P.

We also need the following definitions.

1. If an edge e_i^Q of a circumscribing polygon Q touches P (which it must, if Q has minimal area or perimeter), then its *contact point* is the part of P that it touches. A contact point is always either an edge or a vertex of P.

2. Two circumscribing polygons Q and Q' *interleave* if the contact points of Q and Q' alternate. In other words, between every two consecutive contact points of Q (in the counterclockwise ordering of vertices and edges of P) is a contact point of Q' (perhaps one of the two consecutive contact points of Q). Similarly, between every two consecutive contact points of Q' is a contact point of Q.

3. An edge e_i^Q of Q is *flush* with P if its contact point is an edge. Q itself is flush with P if all its edges are flush with P.

4. An edge e_i^Q of Q is *balanced* if its midpoint lies on P.

5. An edge e_i^Q of Q determines two half-planes; let H_i denote the half-plane that does not contain P. e_i^Q is a *c-edge* if the lines containing its neighbors e_{i-1}^Q and e_{i+1}^Q converge (*i.e.*, intersect) in H_i or are parallel; otherwise, e_i^Q is a *d-edge*. Equivalently, e_i^Q is a c-edge if the sum of the two internal angles of Q corresponding to e_i^Q's endpoints is greater than or equal to π, and e_i^Q is a d-edge if this sum is less than π.

3.1 Area Minimization

Our algorithm for finding a minimum area circumscribing d-gon has two parts. First, we restrict our attention to *flush* circumscribing d-gons and use the techniques of [BDDG85] to find one with minimal area. Then, we use this minimum area flush d-gon (and a lemma due to DePano [DeP87]) to obtain a circumscribing d-gon, possibly not flush, whose area is minimal among *all* circumscribing d-gons.

3.1.1 Finding the Best Flush d-gon

The techniques given by Boyce et al. [BDDG85] for finding a maximum perimeter inscribed d-gon can also be used to find a minimum area flush circumscribing d-gon in $O(dn \lg n + n \lg^2 n)$ time. (This was pointed out in the concluding section of [BDDG85].) Furthermore, the techniques of [AKM*87] reduce the time complexity of this problem to $O(dn + n \lg n)$. For the sake of completeness, we will describe this result, recasting it in terms of multidimensional totally monotone arrays.

For $1 \leq i \leq n$, let r_i^{ccw} be the ray containing e_i^P with v_i^P as its origin, and let r_i^{cw} be the ray containing e_i^P with v_{i+1}^P as its origin. (The superscript of r_i^{ccw} indicates that it is a counterclockwise "extension" of e_i^P, and the superscript of r_i^{cw} indicates that it is a clockwise "extension.") If r_i^{ccw} intersects r_j^{cw}, let $R_{i,j}$ be the region outside P bounded by r_i^{ccw}, r_j^{cw}, and the edges $e_{i+1}^P, \ldots, e_{j-1}^P$ of P (the shaded region in Figure 3.1). Now consider the two-dimensional arrays $W = \{w_{i,j}\}$ and $W' = \{w'_{i,j}\}$, where

$$w_{i,j} = \begin{cases} \text{area}(R_{i,j}) & \text{if } i < j \text{ and } r_i^{\text{ccw}} \text{ intersects } r_j^{\text{cw}}, \text{ and} \\ \infty & \text{otherwise,} \end{cases}$$

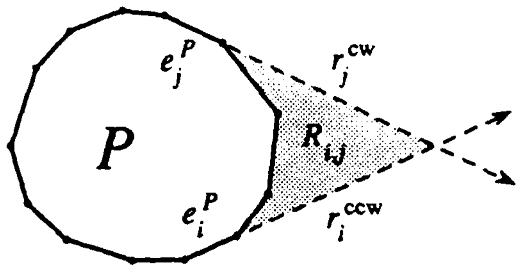

Figure 3.1: If r_i^{ccw} intersects r_j^{cw}, then $R_{i,j}$ is the shaded region between r_i^{ccw}, r_j^{cw}, and P.

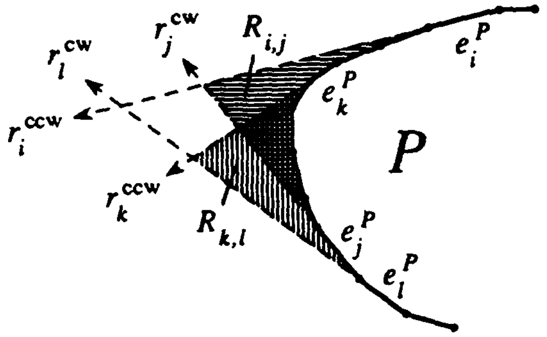

Figure 3.2: Suppose w_{ij}, w_{il}, w_{kj}, and w_{kl} all correspond to regions outside of P, *i.e.*, $i < k < j < l$ and r_i^{ccw} intersects r_l^{cw}. The region R_{il} bounded by r_i^{ccw} and r_l^{cw} contains both the region R_{ij} bounded by r_i^{ccw} and r_j^{cw} (shaded with horizontal lines) and the region R_{kl} bounded by r_k^{ccw} and r_l^{cw} (shaded with vertical lines). Moreover, the intersection of R_{ij} and R_{kl} is exactly the region R_{kj} bounded by r_k^{ccw} and r_j^{cw}.

and

$$w'_{i,j} = \begin{cases} \text{area}(R_{i,j}) & \text{if } i > j \text{ and } r_i^{ccw} \text{ intersects } r_j^{cw}, \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

Lemma 3.1 *Both W and W' satisfy the Monge condition.*

Proof We only prove the lemma for W; the proof for W' is similar. For $1 \leq i < k \leq n$ and $1 \leq j < l \leq n$, we must show that $w_{ij} + w_{kl} \leq w_{il} + w_{kj}$. We consider two cases. If either $w_{il} = \infty$ or $w_{kj} = \infty$, then the Monge condition follows immediately. If, on the other hand, w_{il} and w_{kj} both correspond to regions outside of P (R_{il} and R_{kj}, respectively), then w_{ij} and w_{kl} must also correspond to regions outside of P (R_{ij} and R_{kl}, respectively), and the four regions must overlap as in Figure 3.2. Now consider the region $R_{ij} \cup R_{kl}$. It has area $w_{ij} + w_{kl} - w_{kj}$, since the intersection of R_{ij} and R_{kl} is exactly R_{kj}. Moreover, $R_{ij} \cup R_{kl}$ is contained in R_{il}, which implies $w_{ij} + w_{kl} - w_{kj} \leq w_{il}$ or $w_{ij} + w_{kl} \leq w_{il} + w_{kj}$. ■

By summing d-dimensional extensions of W and W', we obtain the d-dimensional array $A = \{a_{i_1,i_2,\ldots,i_d}\}$, where

$$a_{i_1,i_2,\ldots,i_d} = w_{i_1,i_2} + w_{i_2,i_3} + \cdots + w_{i_{d-1},i_d} + w'_{i_d,i_1}.$$

A is clearly a Monge-composite cycle-decomposable array. Furthermore, A contains an entry corresponding to every possible flush circumscribing d-gon. Specifically, the area of the flush circumscribing d-gon with contact points $e^P_{i_1}, e^P_{i_2}, \ldots, e^P_{i_d}$, $i_1 < i_2 < \cdots < i_d$, is $a_{i_1,i_2,\ldots,i_d} + \text{area}(P)$. Moreover, only those entries corresponding to circumscribing d-gons are less than ∞; thus, to find a minimum area flush circumscribing d-gon, we need only find a maximum entry in A.

By applying Proposition 2.11 directly, we can find this entry in $O(dn \lg n)$ time. However, this time complexity can be reduced to $O(dn + n \lg n)$ using a theorem concerning the area of interleaving d-gons. (The theorem we prove is actually significantly more general than we need it to be in obtaining an $O(dn + n \lg n)$ time algorithm for the minimum area flush circumscribing d-gon problem, but we will use it again later in this section in the context of two other related problems.)

Before we can prove this theorem, however, we first need a few more definitions. Let Q_a and Q_b denote circumscribing d-gons. We define an *edge exchange* operation as follows. Let E denote the union of Q_a and Q_b's edges. To exchange edges between Q_a and Q_b, we select a d-edge subset E' of E, and form two new circumscribing d-gons, the first consisting of the edges of E' (extended or shortened as necessary to form a circumscribing d-gon) and the second consisting of the edges of $E - E'$.

Now let $\mathcal{Q}_a$ and $\mathcal{Q}_b$ denote sets of circumscribing d-gons. We will say that $\mathcal{Q}_a$ and $\mathcal{Q}_b$ are *closed under edge exchange* if for any d-gon $Q_a \in \mathcal{Q}_a$ and any d-gon $Q_b \in \mathcal{Q}_b$, any edge exchange between Q_a and Q_b produces a d-gon in $\mathcal{Q}_a$ and a d-gon in $\mathcal{Q}_b$.

Theorem 3.2 *Suppose two sets $\mathcal{Q}_a$ and $\mathcal{Q}_b$ of circumscribing d-gons are closed under edge exchange. Furthermore, suppose that Q_a has minimum perimeter among d-gons in $\mathcal{Q}_a$ and that Q_b has minimum perimeter among d-gons in $\mathcal{Q}_b$. Then Q_a and Q_b interleave.*

Proof Suppose Q_a and Q_a do not interleave. Let $a_1, a_2, \ldots, a_d$ be the contact points of Q_a, and let $b_1, b_2, \ldots, b_d$ be the contact points of Q_b. Since Q_a and Q_b do not interleave, there exists at least one pair (a_{i-1}, a_i) of consecutive contact points of Q_a such that no contact points of Q_b lie between a_{i-1} and a_i (inclusive) in the counterclockwise ordering of P's vertices and edges. There also exists at least one pair (b_j, b_{j+1}) of consecutive contact points of Q_b such that no contact points of Q_a lie between b_j and b_{j+1} (inclusive). Moreover, there exists such a pair (a_{i-1}, a_i) and such a pair (b_j, b_{j+1}) separately only by alternating contact points. In other words, there exist i, j, and k, such that the contact points between a_{i-1} and b_{j+1} are (in order)

$$a_{i-1}, a_i, \underbrace{b_k, a_{i+1}, b_{k+1}, a_{i+2}, \ldots, b_{j-1}, a_{j-k+i}}_{\textbf{alternating contact points}}, b_j, b_{j+1}.$$

Now suppose we exchange edges between Q_a and Q_b and form a d-gon Q'_a with contact points

$$a_1, \ldots, a_{i-1}, b_k, b_{k+1}, \ldots, b_j, a_{j-k+i+1}, \ldots, a_d$$

and a d-gon Q'_b with contact points

$$b_1, \ldots, b_{k-1}, a_i, a_{i+1}, \ldots, a_{j-k+i}, b_{j+1}, \ldots, b_d.$$

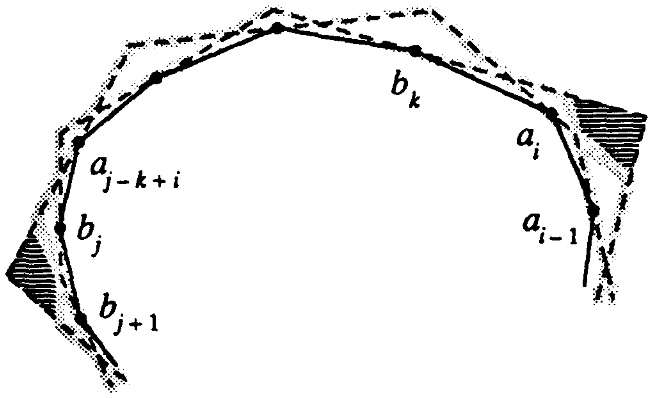

Figure 3.3: The sum of the areas of Q_a and Q_b (indicated by the dotted lines) is exactly the sum of the areas of Q'_a and Q'_b (indicated by the shaded lines) *plus* the area of the two shaded regions.

For any two contact points c and c', let $R_{c,c'}$ denote the region outside P bounded by P and lines through the edges of Q_a or Q_b touching c and c'. As is suggested in Figure 3.3,

$$\begin{aligned}\text{area}(Q'_a) + \text{area}(Q'_b) &= \text{area}(Q_a) + \text{area}(Q_b)\\ &\quad - \text{area}(R_{a_{i-1},a_i}) - \text{area}(R_{a_{j-k+i},a_{j-k+i+1}})\\ &\quad - \text{area}(R_{b_{k-1},b_k}) - \text{area}(R_{b_j,b_{j+1}})\\ &\quad + \text{area}(R_{a_{i-1},b_k}) + \text{area}(R_{b_j,a_{j-k+i+1}})\\ &\quad + \text{area}(R_{b_{k-1},a_i}) + \text{area}(R_{a_{j-k+i},b_{j+1}}).\end{aligned}$$

Since b_{k-1} precedes a_{i-1} in the counterclockwise ordering of P's vertices and edges,

$$\text{area}(R_{b_{k-1},a_i}) + \text{area}(R_{a_{i-1},b_k}) < \text{area}(R_{b_{k-1},b_k}) + \text{area}(R_{a_{i-1},a_i}),$$

by the same argument we used in the proof of Lemma 3.2. Similarly,

$$\text{area}(R_{a_{j-k+i},b_{j+1}}) + \text{area}(R_{b_j,a_{j-k+i+1}}) < \text{area}(R_{a_{j-k+i},a_{j-k+i+1}}) + \text{area}(R_{b_j,a_{j+1}}).$$

Thus, $\text{area}(Q'_a) + \text{area}(Q'_b) < \text{area}(Q_a) + \text{area}(Q_b)$. Since $\mathcal{Q}_a$ and $\mathcal{Q}_b$ are closed under edge exchange, one of the new d-gons is in $\mathcal{Q}_a$ and the other is in $\mathcal{Q}_b$. Without loss of generality, we assume $Q'_a \in \mathcal{Q}_a$ and $Q'_b \in \mathcal{Q}_b$. Now either $\text{area}(Q'_a) < \text{area}(Q_a)$ or $\text{area}(Q'_b) < \text{area}(Q_b)$, both of which are contradictions. ■

Corollary 3.3 *Let Q_i be a minimum area flush circumscribing d-gon with e_i^P as a contact point. Every minimum area flush circumscribing d-gon interleaves Q_i.*

Proof Any edge exchange between a circumscribing d-gon with e_i^P as a contact point and an arbitrary circumscribing d-gon produces a circumscribing d-gon with e_i^P as a contact point and an arbitrary circumscribing d-gon. Thus, this corollary follows from Thereom 3.2. ■

Returning to the problem of finding a minimum area flush circumscribing d-gon, note that finding a minimum area flush circumscribing d-gon Q_1 with e_1^P as a contact point is equivalent to finding a minimum entry in first plane of A. This can be done in $O(dn)$ time (since this plane is path-decomposable). Let $e^P_{j_1}, \ldots, e^P_{j_d}$ be the contact points of Q_1 (by definition, $j_1 = 1$; by convention. $j_{d+1} = n$). These edges define d intervals $I_1, \ldots, I_d$ of edges from P, where $I_k = [e^P_{j_k}, e^P_{j_{k+1}}]$. Corollary 3.3 tells us that we need only consider edges in I_k for the k-th contact point of a minimum area circumscribing d-gon. In other words, we need only search the $n_1 \times n_2 \times \cdots \times n_d$ subarray of A, $n_k = j_{k+1} - j_k + 1$, containing those entries $a_{i_1,i_2,\ldots,i_d}$ where $j_k \leq i_k \leq j_{k+1}$ for all k between 1 and d. Since

$$\sum_{k=1}^{d} n_k = O(n)$$

and every subarray of a cycle-decomposable array is also cycle-decomposable, we can use Theorem 2.11 to find a minimum entry in this subarray, corresponding to a minimum area flush circumscribing d-gon, in $O(n \lg n)$ additional time. This gives the entire algorithm a time complexity of $O(dn + n \lg n)$.

3.1.2 Using the Best Flush d-gon to Obtain the Best Arbitrary d-gon

In [DeP87], DePano provides the following geometric characterization of minimum area circumscribing d-gons.

Lemma 3.4 ([DeP87]) *Let P be any convex n-gon. For $3 \leq d \leq n$, if Q is a minimum area d-gon Q circumscribing P, then either*

1. *all d edges of Q are flush with P, or*

2. *$d - 1$ edges of Q are flush with P, and the non-flush edge is a balanced d-edge.*

This lemma allows us to relate minimum area flush circumscribing d-gons and minimum area arbitrary circumscribing d-gons. Specifically, we have the following corollary to Theorem 3.2.

Corollary 3.5 *Let Q' be a minimum area flush circumscribing d-gon. Every minimum area circumscribing d-gon Q interleaves Q'.*

Proof Let $\mathcal{Q}'$ denote the set of all flush circumscribing d-gons, and let $\mathcal{Q}$ denote the set of all circumscribing d-gons whose first $d-1$ edges are flush with P and whose d-th edge is a balanced d-edge. By Lemma 3.4, every minimum area circumscribing d-gon is in $\mathcal{Q} \cup \mathcal{Q}'$. Moreover, every minimum area circumscribing d-gon has minimal area among d-gons in $\mathcal{Q} \cup \mathcal{Q}'$.

Now, $\mathcal{Q}'$ and $\mathcal{Q} \cup \mathcal{Q}'$ are clearly closed under edge exchange. Thus, by Theorem 3.2, every minimum area circumscribing d-gon must interleave every minimum area flush circumscribing d-gon. ■

Now suppose we have found a minimum area flush circumscribing d-gon Q', using the techniques of [BDDG85]. Let $e^P_{j_1}, \ldots, e^P_{j_d}$ be the contact points of this d-gon. Without loss of generality, assume $j_1 = 1$ (if it is not, we can renumber the edges of P). Also, for notational convenience, let $j_{d+1} = n$. The edges $e^P_{j_1}, \ldots, e^P_{j_d}$ define d intervals $I_1, \ldots, I_d$ of edges from P, where $I_k = [e^P_{j_k}, e^P_{j_{k+1}}]$. Lemma 3.5 tells us that we need only consider edges or vertices in I_k for the k-th contact point of a minimum area circumscribing d-gon Q. (A vertex is considered to be in I_k if *both* the edges incident to it are in I_k.) Furthermore, we can use Lemma 3.4 to show:

Proposition 3.6 *There are at most three intervals that might contain the contact point of the non-flush edge of a minimum area circumscribing d-gon Q. Moreover, we can identify these intervals in $O(d)$ time.*

Proof For each edge e^P_i of P, let r^{ccw}_i be the ray containing e^P_i with v^P_i as its origin. Now for each interval I_k, consider the ray $r^{ccw}_{j_k}$ associated with the interval's first edge $e^P_{j_k}$. Let α_k be $r^{ccw}_{j_k}$'s angle with respect to r^{ccw}_1, measured in a counterclockwise direction. Clearly,

$$0 = \alpha_1 < \alpha_2 < \cdots < \alpha_d < 2\pi.$$

Now suppose the non-flush edge of Q is e^Q_k. It must contain an edge or vertex of P in the interval I_k. Moreover, the contact point of e^Q_{k-1} is an edge $e^P_i \in I_{k-1}$ and the contact point of e^Q_{k+1} is an edge $e^P_{i'} \in I_{k+1}$. Since e^P_i lies in I_{k-1}, the angle r^{ccw}_i forms with r^{ccw}_1 is at least α_{k-1}. Similarly, since $e^P_{i'}$ lies in I_{k+1}, the angle $r^{ccw}_{i'}$ forms with r^{ccw}_1 is at most α_{k+2}. Furthermore, DePano's lemma tells us that e^Q_k is a d-edge, *i.e.*, the lines containing e^Q_{k-1} and e^Q_{k+1} intersect on P's side of the line containing e^Q_k. This implies $(\alpha_{k+2} - \alpha_{k-1}) \bmod 2\pi > \pi$, and this inequality can hold for at most three values of k.

To identify those intervals that might contain the contact point of the non-flush edge of Q, we need only compute α_k for each interval I_k and then $(\alpha_{k+2} - \alpha_{k-1}) \bmod 2\pi$ for each k, which may be done in $O(d)$ time. ■

We can check the possibility of the nonflush edge lying in the interval I_k as follows. Let e^P_i be any edge of P in I_{k-1}. Let e^P_j be any edge of P in I_{k+1}. Define $\mathcal{Q}_{i,j}$ to be the set of all circumscribing d-gons Q satisfying the following constraints:

1. Q's $(k-1)$-st contact point is the edge e^P_i,
2. Q's k-th contact point is an edge or vertex in I_k,
3. Q's k-th edge is a d-edge,
4. Q's $(k+1)$-st contact point is the edge e^P_j, and
5. for $1 \leq l < k-1$ and $k+1 < l \leq d$, Q's l-th contact point is an edge in I_l.

Also define

$$\mathcal{Q}_i = \bigcup_j \mathcal{Q}_{i,j}.$$

Lemma 3.7 *For any $e^P_i \in I_{k-1}$, we can find a circumscribing d-gon $Q_i \in \mathcal{Q}_i$ of minimal area in $O(n)$ time.*

Proof We begin with a few more definitions (borrowed from [ACY85]). An *h-sided* (i, j)-*chain* is a polygonal chain $C = \{e_1^C, \ldots, e_h^C\}$ such that

1. e_1^C is flush with e_i^P,
2. e_h^C is flush with e_j^P, and
3. for $1 < l < h$, e_l^C has a contact point in $I_{(k+l) \bmod d}$.

C is *flush* if all its edges are flush. The *extra area* of C is the area of the bounded (but perhaps disconnected) region between C and P.

For each edge $e_j^P \in I_{k+1}$, let $C_{i,j}$ be an optimal flush $(d-1)$-sided (j, i)-chain (*i.e.*, its extra area is minimal), and let $C'_{i,j}$ be an optimal 3-sided (i, j)-chain. Combining $C_{i,j}$ and $C'_{i,j}$ gives an optimal circumscribing d-gon $Q_{i,j}$ from $\mathcal{Q}_{i,j}$. Moreover, given $Q_{i,j}$ for each $e_j^P \in I_{k+1}$, we can pick the best of these d-gons in $O(n)$ time to obtain an optimal circumscribing d-gon from $\mathcal{Q}_i$. Thus, if we can find an optimal flush $(d-1)$-sided (j, i)-chain and an optimal 3-sided (i, j)-chain for each $e_j^P \in I_{k+1}$ in $O(n)$ time, we will have established the lemma.

To find an optimal flush $(d-1)$-sided (j, i) chain for each $e_j^P \in I_{k+1}$, we first define the $(d-2)$-dimensional path-decomposable array $A = \{a_{i_1, i_2, \ldots, i_{d-2}}\}$, where

$$a_{i_1, i_2, \ldots, i_{d-2}} = w_{i_1, i_2} + w_{i_2, i_3} + \cdots + w_{i_{d-k-1}, i_{d-k}} + w'_{i_{d-k+1}, i_{d-k}} + w_{i_{d-k+1}, d-k+2} + \cdots + w_{i_{d-2}, i}.$$

(We defined the arrays $W = \{w_{i,j}\}$ and $W' = \{w_{i,j}\}$ earlier in this section, in the context of optimal flush circumscribing d-gons.) Now suppose $I_l = [j_l, j_{l+1}]$ for $1 \leq l \leq d$. Let $n_l = j_{l+1} - j_l + 1$, and let A' be the $n_1 \times n_2 \times \cdots \times n_{d-2}$ subarray of A containing entries $a_{i_1, i_2, \ldots, i_{d-2}}$ that satisfy $j_{(l+k) \bmod d} \leq i_l \leq j_{(l+k+1) \bmod d}$ for $1 \leq l \leq d-2$. The optimal flush $(d-1)$-sided (j, i) chain corresponds to the maximum entry in the j-th plane of A' (the plane containing those entries of A' whose first coordinate is j). Since $\sum_{l=1}^{d-2} n_l = O(n)$, Theorem 2.10 tells us that we can compute these plane maxima in $O(n)$ time.

To find an optimal 3-sided (i, j)-chain for each $e_j^P \in I_{k+1}$, we first recall that Lemma 3.4 tells us we need only consider 3-sided chains whose middle edge is a balanced d-edge. We will call this middle edge a *closing edge* for e_i^P and e_j^P.

Now suppose e_k^Q is a balanced closing d-edge for $e_i^P \in I_{k-1}$ and $e_j^P \in I_{k+1}$. e_k^Q's first endpoint must lie on the line L_i containing e_i^P, and its second endpoint must lie on the line L_j containing e_j^P. Moreover, since e_k^Q is balanced, its second endpoint must also lie on the chain C defined as follows. For each $e_l^P \in I_k$, let e_{2l-1}^C be the segment on the line containing e_l^P whose first and second endpoints are twice as far from L_i as are e_l^P's first and second endpoints, respectively. C consists of these segments, plus segments e_{2l}^C parallel to L_i connecting e_{2l-1}^C and e_{2l+1}^C for $1 \leq l < |I_k|$. This is suggested in Figure 3.4. We will denote the endpoints of e_l^C by v_l^C and v_{l+1}^C.

Using what we know about the angles of rays containing $e_l^C \in C$ and $e_j^P \in I_{k+1}$ form with respect r_i^{ccw}, we can prove the following two propositions about C and the lines L_j containing edges in I_{k+1}.

Proposition 3.8 *For all $e_j^P \in I_{k+1}$, L_j intersects C at most once.*

Proposition 3.9 *For all e_j^P and $e_{j'}^P$ in I_{k+1}, $j' > j$, L_j intersects C before $L_{j'}$ does,* i.e., *if L_j intersects e_l^C, then $L_{j'}$ intersects $e_{l'}^C$, $l' > l$.*

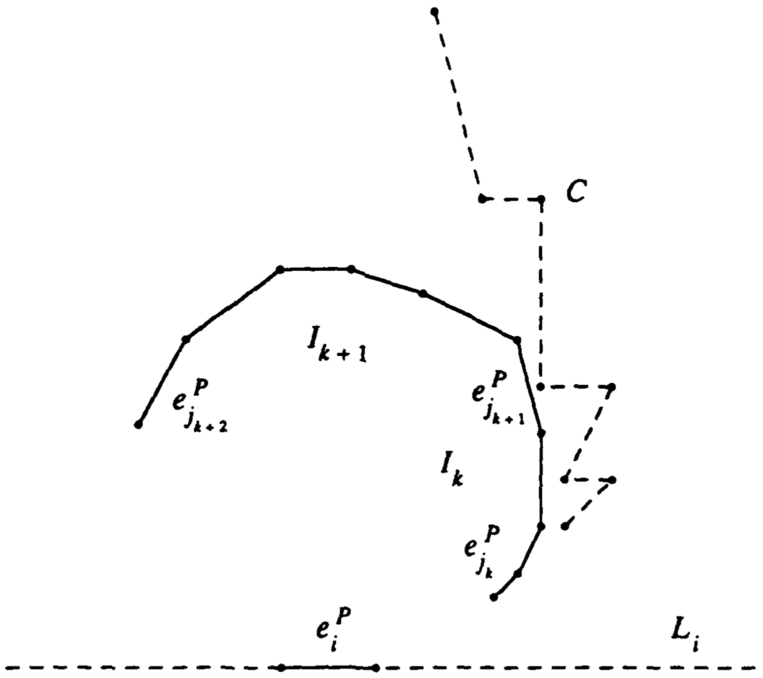

Figure 3.4: Every balanced closing d-edge for a particular $e^P_i \in I_{k-1}$ and any $e^P_j \in I_{k+1}$ must have one endpoint on the chain C.

The first proposition tells us that there is at most one balanced closing d-edge corresponding to each $e^P_j \in I_{k+1}$. The second allows us to find all these closing edges in $O(n)$ time. We begin by computing C, which can be done in $O(n)$ time. We then check whether the line L containing the first edge in I_{k+1} intersects e^C_1. If L does not intersect this segment, we check whether it intersects e^C_2. If again there is no intersection, we move on to e^C_3. We continue in this manner until we find the segment e^C_l that L intersects. This gives us the closing edge for the first edge of I_{k+1} Next, we check whether the line L' containing the second edge of I_{k+1} intersects e^C_l. If there is no intersection, we move on to segment e^C_{l+1}. We continue in this manner until all closing edges are found. It is easy to see that only $O(n)$ time is required. Also, Proposition 3.9 guarantees that this approach will find all of the desired intersections, which in turn gives us the optimal 3-sided (i, j)-chains. ■

To complete our algorithm, we require a third corollary to Theorem 3.2.

Corollary 3.10 *For any e^P_i and $e^P_{i'}$ in I_{k-1}, every optimal $Q_i \in \mathcal{Q}_i$ interleaves every optimal $Q_{i'} \in \mathcal{Q}_{i'}$.*

Proof This corollary follows immediately from Theorem 3.2, since $\mathcal{Q}_i$ and $\mathcal{Q}_{i'}$ are closed under edge exchange. ■

This corollary, together with Lemma 3.7, allows us to find an optimal circumscribing d-gons for each Q_i such that $e^P_i \in I_{k-1}$ in $O(n \lg n)$ total time. We use the natural divide-and-conquer approach of Theorem 2.6. Let $I_l = [j_l, j_{l+1}]$ for $1 \leq l \leq d$, and let $n_l =$

$j_{l+1} - j_l + 1$. We first find an optimal circumscribing d-gon $Q_i \in \mathcal{Q}_i$ for $i = j_{k-1} + \lceil n_{k-1}/2 \rceil$. By Corollary 3.10, the contact point of Q_i in interval I_l, $1 \leq l \leq d$, splits that interval into two intervals I'_l and I''_l, such that the contact points of any optimal circumscribing d-gon for $\mathcal{Q}_{i'}$, $j_{k-1} \leq i' < i$, must lie in the intervals $I'_1, \ldots, I'_d$, and the contact points of any optimal circumscribing d-gon for $\mathcal{Q}_{i'}$, $i < i' < j_k$, must lie in the intervals $I''_1, \ldots, I''_d$. If we recursively solve the two subproblems associated with the intervals $I'_1, \ldots, I'_d$ and the intervals $I''_1, \ldots, I''_d$, we obtain a recurrence with solution $O(n \lg n_{k-1}) = O(n \lg n)$ for the time required to find an optimal circumscribing d-gon for each $\mathcal{Q}_i$ such that $e^P_i \in I_{k-1}$.

By choosing the best of the n_{k-1} circumscribing d-gons obtained in this manner (which can be done in $O(n)$ time), we obtain a minimum area circumscribing d-gon; thus,

Theorem 3.11 *Given a convex n-gon, a minimum area circumscribing d-gon can be computed in $O(dn + n \lg n)$ time.*

3.2 Perimeter Minimization

Just as the techniques given by Boyce et al. [BDDG85] for finding a maximum perimeter inscribed d-gon can be used to find a minimum area flush circumscribing d-gon in $O(dn + n \lg n)$ time, they can also be used to find a minimum perimeter flush circumscribing d-gon in the same amount of time. Unfortunately, we lack a lemma analogous to Lemma 3.4 for the case of perimeter minimization, and thus it is unclear how to use this minimum perimeter flush circumscribing d-gon to obtain a minimum perimeter arbitrary circumscribing d-gon. In fact, the only previous algorithm for the minimum perimeter circumscribing d-gon problem, due to DePano [DeP87], is restricted to the case of $d = 3$. Specifically, DePano obtained an $O(n^3)$ time algorithm for the minimum perimeter circumscribing triangle problem. In doing so, he proved the following two lemmas, which we will use in obtaining an $O(n \lg n)$ time algorithm for the problem.

Lemma 3.12 ([DeP87]) *For every convex n-gon P, there exists a minimum perimeter circumscribing triangle Q with at least one edge flush with P.*

Lemma 3.13 ([DeP87]) *For any triple (i, j, k), there is a unique minimum perimeter circumscribing triangle $T_{i,j,k}$ whose first edge e^T_1 is flush with edge e^P_i of P, whose second edge e^T_2 contains vertex v^P_j of P, and whose third edge e^T_3 contains vertex v^P_k of P. Moreover, this triangle satisfies the following condition: there exists a point p on e^T_2 and P and a point q on e^T_3 and P such that the distance between p and the endpoint shared by e^T_1 and e^T_2 equals the distance between q and the endpoint shared by e^T_1 and e^T_3 (see Figure 3.5). Note that p must be v^P_j if e^T_2 is not flush with P, and q must be v^P_k if e^T_3 is not flush with P.*

Lemma 3.12 tells us that we need only consider circumscribing triangles from the set

$$\mathcal{T} = \{T_{i,j,k} \mid 1 \leq i \leq n, 1 \leq j < k \leq n\},$$

as this set must contain a minimum perimeter circumscribing triangle. For any i, j, and k, Lemma 3.13 allows us to compute $T_{i,j,k}$ (and its perimeter) in constant time (see [DeP87] for details).

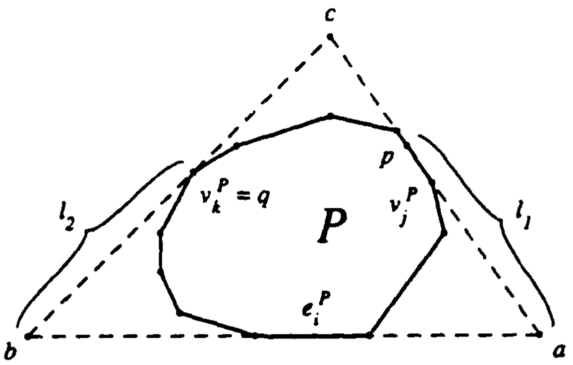

Figure 3.5: If $\triangle abc$ is the minimum perimeter circumscribing triangle containing edge e_i^P and vertices v_j^P and v_k^P of P, then there exist points p and q on both P and $\triangle abc$ such that the distance l_1 between a and p equals the distance l_2 between b and q.

DePano's $O(n^3)$ time algorithm for the minimum circumscribing triangle problem is nothing more than a brute force search through $\mathcal{T}$ — he just computes the perimeter of every triangle in $\mathcal{T}$ and then outputs one with minimal perimeter. We obtain a substantially better algorithm by identifying a multidimensional Monge array associated with $\mathcal{T}$ and applying our array-searching techniques.

Specifically, we define the following $n \times (2n-2) \times (2n-1)$ array $A = \{a_{ijk}\}$. For $1 \leq i \leq n$ and $i < j < k < i+n$, we let a_{ijk} be the perimeter of $T_{i,j,k}$ (which we denote $\text{per}(T_{i,j,k})$), provided this triangle exists (the indices j and k are modulo n). If this triangle does not exist, we let a_{ijk} be ∞. For all other values of i, j, and k, we also define a_{ijk} to be ∞.

Now, every triangle in $\mathcal{T}$ is represented in A, and the only entries in A that are less than ∞ correspond to triangles. Thus, by Lemma 3.12, the minimum entry in A corresponds to the minimum perimeter circumscribing triangle. Moreover, since Lemma 3.13 implies each entry of A can be computed in constant time, the following lemma gives us an $O(n \lg n)$ time algorithm for computing a minimum perimeter circumscribing triangle.

Lemma 3.14 *A satisfies the Monge condition.*

Proof Consider any two-dimensional plane of A corresponding to a fixed value i of A's first coordinate. Suppose this plane does not satisfy the Monge condition, *i.e.*, there exist $j < j'$ and $k < k'$ such that

$$a_{ijk} + a_{ij'k'} > a_{ijk'} + a_{ij'k}. \tag{3.1}$$

We cannot have $j' \geq k$, since this would mean $a_{ij'k} = \infty$, which contradicts (3.1). Similarly, $T_{ijk'}$ must exist, which implies T_{ijk}, $T_{ij'k'}$, and $T_{ij'k}$ also exist. Now suppose we exchange edges between $T_{ijk'}$ and $T_{ij'k}$. Specifically, we let T'_{ijk} be the triangle formed by the first and second edges of $T_{ijk'}$ and the third edge of $T_{ij'k}$ (extended or shortened, as necessary, to form a triangle), and let $T'_{ij'k'}$ be the triangle formed by the first and second edges of $T_{ij'k}$ and the third edge of $T_{ijk'}$. This is suggested in Figure 3.6. Now it is easy to verify that

$$\text{per}(T'_{ijk}) + \text{per}(T'_{ij'k'}) \leq \text{per}(T_{ijk'}) + \text{per}(T_{ij'k}).$$

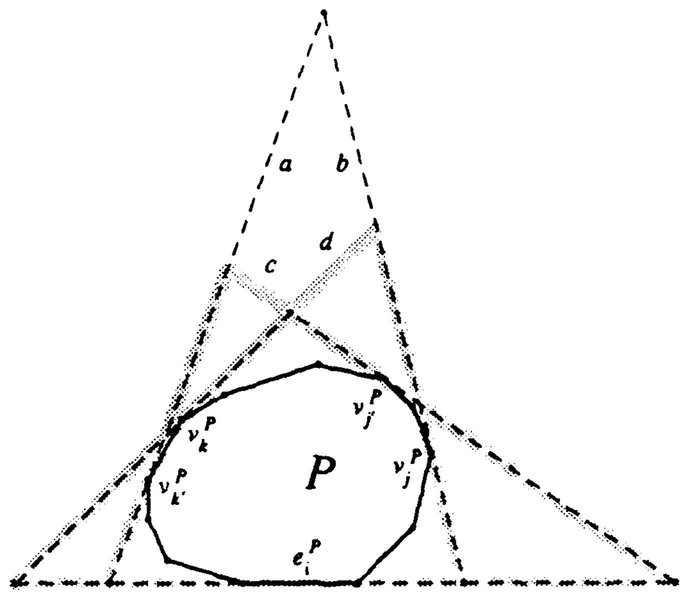

Figure 3.6: Suppose we exchange edges between the triangles $T_{ijk'}$ and $T_{ij'k}$ (indicated by the dotted lines) and obtain the triangles T'_{ijk} and $T'_{ij'k'}$ (indicated by the shaded lines). Then $\text{per}(T'_{ijk}) + \text{per}(T'_{ij'k'}) \leq \text{per}(T_{ijk'}) + \text{per}(T_{ij'k})$, since the length of the line segment a plus the length of the line segment b is at least the length of the line segment b plus the length of the line segment c.

Furthermore, $\text{per}(T_{ijk}) \leq \text{per}(T'_{ijk})$ and $\text{per}(T_{ij'k'}) \leq \text{per}(T'_{ij'k'})$, since T_{ijk} and $T_{ij'k'}$ have minimal perimeter among circumscribing triangles that share their contact points. Thus,

$$\text{per}(T_{ijk}) + \text{per}(T_{ij'k'}) \leq \text{per}(T_{ijk'}) + \text{per}(T_{ij'k}),$$

which contradicts (3.1).

In an similar fashion, we can show that every two-dimensional plane of A corresponding to a fixed value of the second coordinate or a fixed value of the third coordinate also satisfies the Monge condition. ■

It is unclear whether A is Monge-composite, since fixing any two indices of A does not fix the both the edges of the circumscribing triangle corresponding to these indices. Nevertheless, Lemma 3.14 allows us to apply the array-searching algorithm given in Theorem 2.6 and obtain:

Theorem 3.15 *Given a convex n-gon P, a minimum perimeter triangle Q circumscribing P can be computed in $O(n \lg n)$ time.*

It remains open whether a minimum perimeter circumscribing triangle can be found in $o(n \lg n)$ time and whether there exist efficient algorithms for computing minimum perimeter circumscribing d-gons.

4 Dynamic Programming Using Monge Conditions

4.1 Frances Yao's Paper Revisited

In [Yao80], Yao considered the following dynamic programming problem. Let $w(i,j)$ be a weight function defined for $1 \leq i \leq j \leq n$ and satisfying the following two constraints: for $1 \leq i \leq i' \leq j \leq j' \leq n$,

$$w(i,j) + w(i',j') \leq w(i,j') + w(i',j) \tag{4.1}$$

and

$$w(i',j) \leq w(i,j'). \tag{4.2}$$

(Yao called the first constraint the quadrangle inequality and referred to functions satisfying the second constraint as monotone on the lattice of intervals, ordered by inclusion.) We want to compute $c(i,j)$ for $1 \leq i \leq j \leq n$, where $c(i,i) = 0$ for $1 \leq i \leq n$ and

$$c(i,j) \;=\; w(i,j) + \min_{i<k\leq j} \{c(i,k-1) + c(k,j)\}$$

for $1 \leq i < j \leq n$. In [Yao80], Yao showed that this problem can be solved in $O(n^2)$ time. She used this result to provide a simple alternate solution to Knuth's problem regarding optimal binary trees [Knu73]: she also extended this approach to the computation of optimal t-ary trees.

In this subsection, we reformulate Yao's dynamic programming problem in terms of multidimensional Monge arrays and then use a result due to Wilber [Wil88], to obtain an alternate $O(n^2)$ time solution for the problem. We begin with a lemma proved by Yao in obtaining her $O(n^2)$ time bound.

Lemma 4.1 **([Yao80])** *The cost function $c(i,j)$ satisfies the quadrangle inequality,* i.e., $c(i,j) + c(i',j') \leq c(i,j') + c(i',j)$ *for* $1 \leq i \leq i' \leq j \leq j' \leq n$.

Yao's quadrangle inequality is precisely the Monge condition, except that the functions $w(i,j)$ and $c(i,j)$ do not correspond to complete arrays. However, if we let $W = \{w_{i,j}\}$ denote the $n \times n$ array where

$$w_{i,j} \;=\; \begin{cases} w(i,j) & \text{if } i \leq j, \\ \infty & \text{otherwise,} \end{cases}$$

and we let $C = \{c_{i,j}\}$ denote the $n \times n$ array where

$$c_{i,j} \;=\; \begin{cases} c(i,j) & \text{if } i \leq j, \\ \infty & \text{otherwise,} \end{cases}$$

then both W and C satisfy the Monge condition. (Note that it is important that we define $w_{i,j}$ and $c_{i,j}$ to be ∞ when $i > j$ — if we instead define $w_{i,j}$ and $c_{i,j}$ to be $-\infty$ when $i > j$, as we might want to do if we were maximizing instead of minimizing, then W and C would not satisfy the Monge condition or the inverse Monge condition.) Yao's dynamic programming problem then boils down to computing the entries of C. Unfortunately, the results of [AKM*87] seems inapplicable at this point, at least in a straightforward manner, as we are neither interested in the row minima of C nor are the entries of C readily available.

Now consider the three-dimensional array $D = \{d_{i,k,j}\}$, where we define $d_{i,k,j} = c_{i,k-1} + c_{k,j} + w_{i,j}$. Since we now have $c_{i,j} = \min_{1 \le k \le n} d_{i,k,j}$, we have transformed the problem of computing the entries of C to the problem of computing the tube minima of D. Moreover, D is clearly a path-decomposable Monge-composite array (which means it satisfies the Monge condition), but again the array's entries are not directly available. At this point, however, we can apply the results of Wilber [Wil88].

Wilber considered the concave least-weight subsequence problem: given a weight function $v(p,q)$ defined for $0 \le p \le q \le m$ and satisfying what Yao calls the quadrangle inequality, compute $f(q)$ for $0 \le q \le m$, where $f(0) = 0$ and $f(q) = \min_{0 \le p < q}\{f(p) + v(p,q)\}$ for $1 \le q \le m$. We can convert this problem to an array-searching problem by letting $V = \{v_{p,q}\}$ denote the $(m+1) \times (m+1)$ array where

$$v_{p,q} = \begin{cases} v(p,q) & \text{if } p \le q, \\ \infty & \text{otherwise,} \end{cases}$$

and letting $E = \{e_{p,q}\}$ denote the $(m+1)\times(m+1)$ array where $e_{p,q} = f(p)+v_{p,q}$. The concave least-weight subsequence problem is now the column minima problem for E. Moreover, since $v(p,q)$ satisfies the quadrangle inequality, V satisfies the Monge condition, which implies E satisfies the Monge condition. In [Wil88], Wilber showed that this problem can be solved in $O(n)$ time, even though computing an arbitrary entry of E is not a constant time operation.

Returning to Yao's dynamic programming problem, consider the two-dimensional plane of D corresponding to a particular value $\hat{\imath}$ of the first coordinate. The tube minima of D lying in this plane are just the column minima of this plane. Moreover, the column minima problem for the plane is an instance of the concave least-weight subsequence problem. In particular, $m = n - \hat{\imath}$, $v_{p,q} = c_{p+\hat{\imath}+1,q+\hat{\imath}} + w_{\hat{\imath},q+\hat{\imath}}$, and $f(q) = c_{\hat{\imath},q+\hat{\imath}}$. Note that the array V satisfies the Monge condition because the array C satisfies the Monge condition. Provided $c_{i,j}$ is known for $i > \hat{\imath}$ (so that $v_{p,q}$ can always be computed in constant time), we can solve this problem in $O(m)$ time using the algorithm given by Wilber in [Wil88]. Thus, to compute the tube minima of D, we need only apply Wilber's algorithm n times, first to the plane corresponding to $\hat{\imath} = n$, then to the plane corresponding to $\hat{\imath} = n - 1$, and so on down to the plane corresponding to $\hat{\imath} = 1$. This gives the following theorem.

Theorem 4.2 *If the weight function $w(i,j)$ satisfies (4.1) and (4.2), then Yao's dynamic programming problem can be solved in $O(n^2)$ time.*

Now suppose the weight function $w(i,j)$ satisfies

$$w(i,j) + w(i',j') \ge w(i,j') + w(i',j) \tag{4.3}$$

and

$$w(i',j) \ge w(i,j') \tag{4.4}$$

for $1 \le i \le i' \le j \le j' \le n$. By adopting the procedure given above, we can obtain the following theorem.

Theorem 4.3 *If the weight function $w(i,j)$ satisfies (4.3) and (4.4), then Yao's dynamic programming problem can be solved in $O(n^2\alpha(n))$ time, where $\alpha(n)$ denotes the inverse Ackermann function.*

Proof This proof is identical to that given above except that now the arrays W, C, and D defined above are no longer Monge (or even totally monotone). The column minima problem for the plane of D corresponding to a particular value $\hat{\imath}$ of the first coordinate is now equivalent to the *convex* least-weight subsequence problem considered in [EGG88]: given a weight function $v(p,q)$ defined for $0 \le p \le q \le m$ such that $v(p,q) + v(p',q') \ge v(p,q') + v(p',q)$ for $0 \le p \le p' \le q \le q' \le m$, compute $f(q)$ for $0 \le q \le m$, where $f(0) = 0$ and $f(q) = \min_{0 \le p < q}\{f(p) + v(p,q)\}$ for $1 \le q \le m$. Klawe and Kleitman [KK88] have shown that this problem can be solved in $O(n\alpha(n))$ time. Thus, by applying Klawe and Kleitman's algorithm n times, we can obtain the entries of C in $O(n^2\alpha(n))$ time. ∎

It remains open whether the time complexity given in Theorem 4.3 can be improved from $O(n^2\alpha(n))$ to $O(n^2)$.

4.2 Waterman's Problem

The *primary structure* of a single-stranded RNA molecule is the sequence of ribonucleotides or bases making up the molecule. When the primary structure of an RNA molecule is known, the question of which bases form pairs (thus pinching off loops in the chain of bases) becomes important — this structure is referred to as the *secondary structure* of the RNA. In [Wat78], Waterman argued that predicting the secondary structure of an RNA molecule from its primary structure is closely related to solving some dynamic programming problems. Among the dynamic programming problems he described is the following: given a weight function $w(i,j)$ defined for $1 \le i \le j \le n$ and satisfying the quadrangle inequality (*i.e.*,

$$w(k,l) + w(p,q) \le w(k,q) + w(p,l) \tag{4.5}$$

for $1 \le k \le p \le l \le q \le n$), compute

$$e(i,j) = \min_{\substack{1 \le i' < i \\ 1 \le j' < j}} \{c(i',j') + w(i'+j', i+j)\}$$

for all i and j between 1 and n, where $e(i,1)$ and $e(1,j)$ are given for all i and j and $c(i,j)$ can be computed from $e(i,j)$ in constant time.

The naive approach gives an $O(n^4)$ time algorithm for this dynamic programming problem. Waterman and Smith improved this time bound to $O(n^3)$ in [WS86], and then recently Eppstein, Galil, and Giancarlo [EGG88] obtained an $O(n^2 \lg^2 n)$ time algorithm for the problem. (They also obtained an $O(n^2 \lg n \lg\lg n)$ time algorithm for the special case of w a logarithm or other simple function.) Using array-searching, we show:

Theorem 4.4 *If the weight function $w(i,j)$ satisfies the quadrangle inequality (4.5), then Waterman's dynamic programming problem can be solved in $O(n^2 \lg n)$ time.*

Proof We only prove the theorem for the case $c(i,j) = e(i,j)$, but the proof is easily generalized to any $c(i,j)$ that can be computed from $e(i,j)$ in constant time.

We begin by extending the function $w(i,j)$ to form an $n \times n$ array, just as we did in the context of Yao's dynamic programming problem. In particular, we let $W = \{w_{i,j}\}$ denote the array where

$$w_{i,j} = \begin{cases} w(i,j) & \text{if } i \leq j, \\ \infty & \text{otherwise.} \end{cases}$$

As before, this array satisfies the Monge condition. We will also view the function $e(i,j)$ as an $n \times n$ array $E = \{e_{i,j}\}$ where $e_{i,j} = e(i,j)$, so that the problem is now that of computing the entries of E.

We now require a few definitions and observations. We will say that an entry $e_{i,j}$ from E *strictly dominates* an entry $e_{i',j'}$ if $i > i'$ and $j > j'$. Note that an entry of E depends only on those entries it strictly dominates. We will also define the *k-th diagonal* of E to consist of those entries $e_{i,j}$ such that $i + j = k$. Note that the only entry on the k-th diagonal of E that we need to consider in computing $e_{i,j}$ is the minimum entry $e_{i',j'}$ on the diagonal that is strictly dominated by $e_{i,j}$. We will refer to this entry as a *diagonal minimum*.

With this is mind, we define

$$d_k(i,j) = \min_{\substack{i'<i \\ j'<j \\ i'+j'=k}} e_{i',j'}.$$

(If $k = 1$ or $k > i + j - 2$, we let $d_k(i,j) = \infty$.) We then have

$$e_{i,j} = \min_{1 \leq k \leq 2n} \{d_k(i,j) + w_{k,i+j}\}.$$

We can now describe our algorithm for computing the entries of E. We use a divide-and-conquer approach, reminiscent of an algorithm given by Aggarwal and Suri [AS87] for finding the largest empty corner rectangle. Without loss of generality, we assume n is a power of 2. We begin by partitioning E into four $n/2 \times n/2$ subarrays A, B, C, and D. A contains entries from the first $n/2$ rows and first $n/2$ columns of E, B contains entries from the first $n/2$ rows and last $n/2$ columns, C contains entries form the last $n/2$ rows and first $n/2$ columns, and D contains the remaining entries. We then recursively compute the entries of A; this requires $T(n/2)$ time, where $T(n)$ is the time required to compute the entries of an $n \times n$ array. Next, we compute the *effect* of A on B, *i.e.*, for each entry of B, we obtain an upper bound on its value based on the entries of A. (Each entry in B is the minimum of a number of terms; some of these terms depend only on entries from B; the rest depend only on entries from A; we compute the minimum of those terms dependent on entries from A.) We will explain later how this may be accomplished in $O(n^2)$ time. We then recursively compute the entries of B in $T(n/2)$ time, taking into account the effect of A on B, *i.e.*, in assigning an entry of B its value, we always take the minimum of the value we have recursively computed for this entry based on the entries of B, and the value we have computed for this entry based on the entries of A. We repeat this process for C, computing the effect of A on C in $O(n^2)$ time and then recursively computing C in $T(n/2)$ time. Finally, we compute the effect of A on D, the effect of B on D, and the effect of C on D, all in $O(n^2)$ time, and then recursively compute D in $T(n/2)$ time.

This yields the recurrence $T(n) \leq 4T(n/2) + O(n^2)$ for the time to compute all entries of an $n \times n$ array. Since $T(1) = O(1)$, $T(n) = O(n^2 \lg n)$.

We can compute the effect of A on B as follows (computing the effect of A on C, A on D, B on D, and C on D may be done in an analogous manner). For $1 \leq i \leq n/2$, each entry $e_{i,j+n/2}$ in row i of B strictly dominates the same set of entries in A and thus depends on the same diagonal minima. This motivates defining the $(i + n/2) \times n/2$ array $X^i = \{x^i_{k,j}\}$ where $x^i_{k,j} = d_k(i, 1 + n/2) + w_{k,i+j+n/2}$. $e_{i,j+n/2}$ can then be expressed as the minimum of the following two minima:

$$\min_{1 \leq k \leq i+n/2} x^i_{k,j}$$

and

$$\min_{\substack{1 \leq i' < i \\ 1 \leq j' < j}} \{e_{i',j'+n/2} + w_{i'+j'+n/2,i+j+n/2}\}.$$

As the first of these quantities depends only on entries in A and the second depends only on entries in B, computing the effect of A on B reduces to computing the column minima in all n of the X^i. Furthermore, observe that for all i between 1 and $n/2$, the array X^i satisfies the Monge condition, since W satisfies the Monge condition. Thus, we have the following lemma.

Lemma 4.5 *The column minima of X^i for all i between 1 and $n/2$ can be computed in $O(n^2)$ time.*

Proof of Lemma 4.5 The $d_k(i, n/2)$ for $1 \leq i \leq n/2$ and $1 \leq k \leq n$ can be computed in $O(n^2)$ time, since $d_k(1, n/2) = \infty$ for $1 \leq k \leq n$,

$$d_k(i,n/2) = \begin{cases} \min\{d_k(i-1,n/2), e_{i-1,k-i+1}\} & \text{if } i \leq k \leq n/2 + i - 2, \\ d_k(i-1,n/2) & \text{otherwise,} \end{cases}$$

for $2 \leq i \leq n/2$ and $1 \leq k \leq n$, and $e_{i,j}$ is known for i and j between 1 and $n/2$. Once this has been done, any $x^i_{k,j}$ can be computed in constant time, which means we can apply the algorithm of [AKM*87] n times and obtain all the desired column minima in an additional $O(n^2)$ time. ■

This completes our proof of Theorem 4.4. ■

In [EGG88], Eppstein, Galil, and Giancarlo also considered the variant of Waterman's problem in which the weight function $w(i,j)$ satisfies the inverse quadrangle inequality, *i.e.*,

$$w(k,l) + w(p,q) \geq w(k,q) + w(p,l) \qquad (4.6)$$

for $k \leq p \leq l \leq q$). They obtained an $O(n^2 \lg^2 n)$ time algorithm for this problem using the same techniques they employed for weight functions satisfying the quadrangle inequality. We can apply our techniques to this variant of Waterman's problem, too, and obtain the following theorem.

Theorem 4.6 *If the weight function $w(i,j)$ satisfies the inverse quadrangle inequality (4.6), then Waterman's dynamic programming problem can be solved in $O(n^2 \lg n)$ time.*

Proof This proof is identical to that given above for weight functions satisfying the quadrangle inequality, except that the array W defined above is no longer Monge (or even totally monotone). However, the arrays X^i are Monge (they satisfy the inverse Monge condition), since they are formed from subarrays of W that are Monge. Thus, we can apply the algorithm of [AKM*87] as before. ■

Note that the divide-and-conquer approach our algorithms employ can be used to obtain an $O(n \lg n)$ time solution to the one-dimensional version of Waterman's problem, namely, the concave or convex least-weight subsequence problem considered by [HL87, KK88, Wil88]. However, Wilber [Wil88] solves the concave least-weight subsequence problem in $O(n)$ time by an elegant modification of [AKM*87], and Klawe and Kleitman [KK88] solve the convex least-weight subsequence problem in $O(n\alpha(n))$ time. This suggests that perhaps an $o(n^2 \lg n)$ solution to Waterman's problem might be obtainable.

5 VLSI River Routing

In the section, we apply array-searching to a number of problems that arise in VLSI river routing. These problems involve two pieces (or *modules*) P and Q of a VLSI circuit that need to be wired together. We model P as a sequence of n points (or *terminals*) lying on a horizontal line in the plane. We let p_i denote both the i-th terminal of P and the terminal's distance from p_1. We also assume $p_1 < p_2 < \cdots < p_n$. Similarly, we model Q as another sequence of n terminals lying on a second horizontal line, where q_i denotes both the i-th terminal of Q and the terminal's distance from q_1, and $q_1 < q_2 < \cdots < q_n$. A *placement* for Q (relative to P) consists of an *offset* and a *separation*, which give q_1's horizontal and vertical position, respectively, relative to p_1. Given a placement for Q, a *routing* is a set of n wires (represented by continuous curves), where the i-th wire connects p_i to q_i. In river routing (as opposed to more general wiring models), these wires must be nonintersecting. Furthermore, the wires are constrained to satisfy a set of technology-dependent *design rules.* At a minimum, the distance between any two wires is always at least some fixed constant, which we take to be 1. Examples of other constraints include requiring that the wires are rectilinear, *i.e.*, they lie on a rectilinear integer grid (as they do in Figure 5.1), or that they consist of straight line segments whose orientations with respect to the x-axis are multiples of $\pi/4$. We say a routing is *valid* (for a particular set of design rules) if it satisfies the design rules.

We are concerned with the following three placement problems:

1. *Minimum Separation Problem.* Given a set of design rules and a fixed offset, find the minimum separation that allows a valid routing.

2. *Offset Range Problem.* Given a set of design rules and a fixed separation, find all offsets that allow a valid routing.

3. *Optimal Offset Problem.* Given a set of design rules, find an offset minimizing the separation necessary for a valid routing.

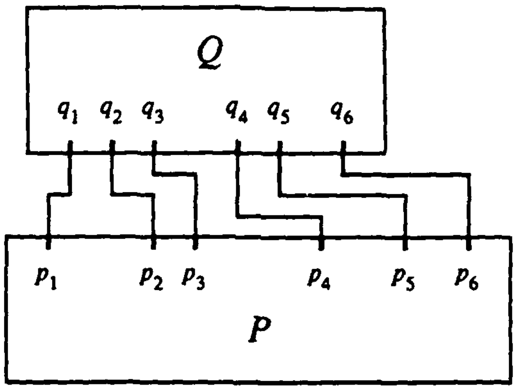
Q
q_1 q_2 q_3 q_4 q_5 q_6
p_1 p_2 p_3 p_4 p_5 p_6
P

Figure 5.1: A rectilinear routing connecting the modules P and Q.

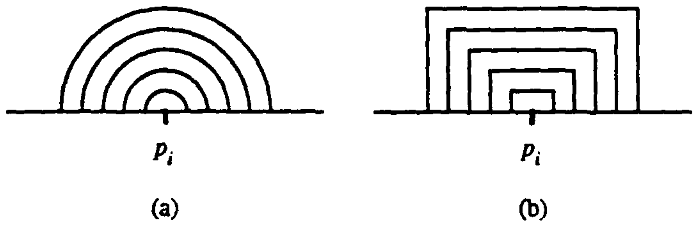

Figure 5.2: The first five barriers about p_i when (a) arbitrary wires are allowed and (b) only rectilinear wires are permitted.

Before we can relate these problems to array-searching, however, we need a few more definitions.

We say that the terminals of P and Q are monotone if $p_i \leq q_i$ for all i or if $p_i \geq q_i$ for all i. For any P and Q, we can assume that the terminals are monotone, since we can always partition the terminals into maximal monotone blocks and find a routing for each of the blocks independently. Without loss of generality, we assume $p_i \leq q_i$ for all i. Since the terminals are monotone. we can assume that the wires are also monotone, *i.e.*, the y-coordinate of a wire is nonincreasing as the x-coordinate increases.

For $1 \leq i \leq n$ and $1 \leq k \leq n-i$, we define the k-th *barrier* about p_i to be the set of points that delimit the closest possible approach to p_i of the monotone wire going from p_{i+k} to q_{i+k}. The barriers are determined by the design rules. For example, if the only design constraint is the lower bound on the distance between wires (the case consider by Tompa in [Tom80]), then the barriers are composed of circular arcs and line segments, as in Figure 5.2(a). If, on the other hand, the design rules allow only rectilinear wires, then the barriers are rectilinear, as in Figure 5.2(b).

Now consider the following $n \times n$ array $A = \{a_{i,j}\}$. If $i < j$, then $a_{i,j}$ is the height at q_j of the $(j-i)$-th barrier about p_i, as in Figure 5.3. If $i \geq j$, then $a_{i,j} = 0$. For $i < j$, the value $a_{i,j}$ is the minimum separation possible for P and Q given the interaction of the terminal p_i with the wire running from p_j to q_j.

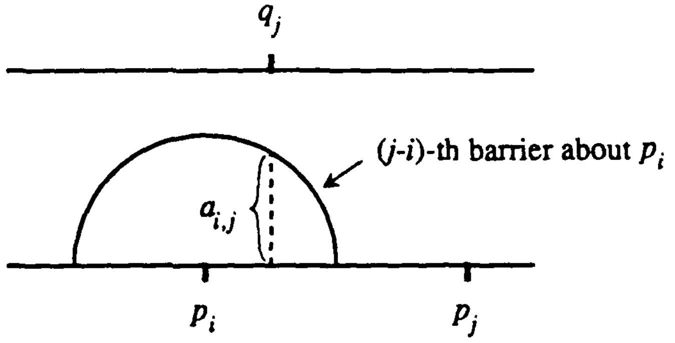

Figure 5.3: The interaction of p_i with the wire running from p_j to q_j requires P and Q be separated by at least $a_{i,j}$.

In [SD81], Siegel and Dolev showed that the minimum separation possible for P and Q is simply the maximum entry in A. They also showed that under some very general assumptions about barriers, corresponding to a wide variety of design rules, the array A is totally monotone. (In fact, their proof is easily extended to show A satisfies the Monge condition.) Under the assumption that each entry of A can be computed in constant time (which is true in most practical applications), Siegel and Dolev then gave an $O(n \lg n)$ time algorithm for the minimum separation problem. Aggarwal et al. [AKM*87] reduced this to $O(n)$ time by simplying applying their linear time algorithm for computing the row maxima of a totally monotone array.

In this paper, we apply array-searching to the offset range and optimal offset problems. We first show that the offset range problem can be solved in linear time for most design rules that occur in practice. In particular, we consider design rules inducing what Siegel and Dolev refer to in [SD81] as a family of similar concave barriers. Such a family of barriers is characterized by a function $h(x)$, where for $0 \leq x < 1$, $0 < h(x) \leq 1$ and $h(x)$ is concave, and for $x \geq 1$, $h(x) = 0$. The height of the k-th barrier about a terminal p_i at a horizontal distance of x from p_i is then given by $kh(x/k)$.

Given a fixed separation w, define the *left offset function* $L(i,j)$ and the *right offset function* $R(i,j)$ as follows: for $1 \leq i \leq n$ and $1 \leq j \leq n$,

$$L(i,j) = \begin{cases} p_i - q_j + (j-i)h^{-1}(w/(j-i)) & \text{if } j-i > w \\ -\infty & \text{if } j-i \leq w \end{cases}$$

and

$$R(i,j) = \begin{cases} p_i - q_j - (i-j)h^{-1}(w/(i-j)) & \text{if } i-j > w \\ \infty & \text{if } i-j \leq w \end{cases} .$$

Siegel and Dolev showed that the offset range for Q is

$$\left[\max_{1\leq i,j\leq n} L(i,j), \min_{1\leq i,j\leq n} R(i,j)\right].$$

They also proved the following theorem.

Theorem 5.1 ([SD81]) *Suppose $r \geq 0$, $s \geq 0$, and $j \geq i + r$. If $L(i,j) \leq L(i+r,j)$, then $L(i,j+s) \leq L(i+r,j+s)$. Similarly, if $R(i,j) \geq R(i+r,j)$, then $R(i,j+s) \geq R(i+r,j+s)$.*

Now let $L = \{l_{i,j}\}$ denote the $n \times n$ array where $l_{i,j} = L(j,i)$ and let $R = \{r_{i,j}\}$ denote the $n \times n$ array where $r_{i,j} = R(j,i)$. An immediate corollary of Theorem 5.1 is the following.

Corollary 5.2 *The arrays L and R are totally monotone. Furthermore, if $L(i,j)$ and $R(i,j)$ can be computed in constant time, then the offset range problem can be solved in linear time by computing the row maxima of L and the row minima of R.*

Proof Consider any 2×2 minor of L, corresponding to rows i and k, $i < k$, and columns j and l, $j < l$. If $l \geq i$, then $l_{i,l} = L(l,i) = -\infty$, which implies the minor is monotone. If, on the other hand, $l < i$, then by Theorem 5.1, $l_{i,j} \leq l_{i,l}$ implies $l_{k,j} \leq l_{k,l}$, which again means the minor is monotone. Thus, L is totally monotone. We can show R is totally monotone in a similar manner.

If we can compute any entry of L or R in constant time, then we can apply the results of [AKM*87] and compute L's row maxima and R's row minima in linear time. We can then compute the maximum of L's row maxima and the minimum of R's row minima in linear time to obtain the offset range for Q. ■

This corollary generalizes the results of Siegel [Sie88], who gave an $O(n)$ time algorithm for the offset range problem when barriers are polygonal in nature.

Passing now to the optimal offset problem, Siegel and Dolev showed in [SD81] that this problem can be solved for a particular set of design rules in $O(\Psi(n) \lg n)$ time if the minimum separation problem can be solved for these design rules in $O(\Psi(n))$ time. Consequently, Corollary 5.2 gives us an $O(n \lg n)$ time algorithm for the optimal offset problem for most design rules used in practice. Though it remains open whether an $o(n \lg n)$ time algorithm for this problem can be obtained for the class of design rules we consider, Mirazaian [Mir87] has provided an elegant $\Theta(n)$ time algorithm for the case of rectilinear wiring; however, his algorithm exploits the properties associated with such a wiring, and it is unlikely that his techniques can be extended to other wiring models.

6 Open Problems

In this paper, we present a framework that allows us to obtain efficient algorithms for wide variety of problems involving quadrangle inequalities. We leave a large number of problems unresolved, however. These open problems may be divided into two types: those relating directly to array-searching and those involving applications of array-searching.

Among the open problem relating directly to array-searching are the following:

1. In Subsection 2.1, we presented an $O(n \lg n)$ time algorithm for computing the plane maxima of an $n \times n \times n$ three-dimensional totally monotone array. This is the best upper bound we have on the time necessary to solve this problem, even for the special case of Monge-composite arrays, whereas the only lower bound known for either of these problems is $\Omega(n)$. Note that our algorithms for computing the plane maxima

of a three-dimensional array A only make use of the monotonicity of A and the total monotonicity of its planes. It is conceivable that an algorithm that takes advantage of the stronger Monge condition might be able to solve the plane maxima problem for A in $o(n \lg n)$ time.

2. In a similar vein, the only upper bound known for the time necessary to compute the plane maxima of an $n \times n \times \cdots \times n$ d-dimensional totally monotone array is $O(n \lg^{d-2} n)$, whereas $\Omega(dn)$ remains the only lower bound known for this problem. Reducing this upper bound to something polynomial in d would be a major improvement. Again, it is conceivable that the stronger Monge condition might be of use in this regard.

3. In [AK88], Aggarwal and Klawe defined an $n \times n$ *staircase-monotone* array to be a totally monotone array with the entries in some contiguous region of the array replaced by $-\infty$'s, so that the array is no longer totally monotone. (This region is constrained to lie in the upper left corner of the array and to be bounded by a staircase growing up and to the right.) Aggarwal and Klawe showed that the row maxima problem for such an array can be solved in $O(n \lg \lg n)$ time and gave a number of applications in which such arrays arise. Klawe and Kleitman [KK88] improved the time bound on this problem to $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. The only lower bound known for this problem is $\Omega(n)$.

As for the applications of array-searching we present in Sections 3–5, we list the open problems in these areas in their respective sections. Clearly, reducing any of the upper bounds given for the open problems listed above will result in the improvement of the corresponding application algorithms, but it is quite possible that exploiting additional properties associated with the individual application problems may help in reducing their computational complexity as well.

In addition to the applications that we describe in this paper, there are also a number of problems that *seem* closely related to the notion of array-searching. Recall that Frances Yao's dynamic programming algorithm requires that the weight function $w(i,j)$ satisfy the quadrangle inequality *and* an additional monotonicity constraint (namely, $w(i',j') \leq w(i,j)$ when $i \leq i' \leq j' \leq j$). Our algorithm for Yao's problem, given in Subsection 4.1, also requires that the weight function satisfy this monotonicity constraint. This constraint seems somewhat restrictive; there are a number of problems, quite similar to Yao's dynamic programming problem, that involve weight functions satisfying the quadrangle inequality but not the additional monotonicity constraint. For example, Gilbert [Gil79] has shown that a minimum weight triangulation of a convex n-gon can be found in $O(n^3)$ time using a dynamic programming algorithm. Though the lengths of the diagonals added in triangulating a convex n-gon do satisfy the quadrangle inequality, they do not obey Yao's monotonicity constraint, thus neither Yao's techniques nor our own are directly applicable. Furthermore, by incorporating the notion of optimal t-ary trees, Gilbert's algorithm can be extended to obtain an $O(n^3 \lg m)$ time algorithm for the problem of partitioning a convex n-gon into convex m-gons such that the sum of the perimeters of these m-gons is minimized [LLS87]. However, again, despite the similarity with Yao's dynamic programming problem, no $o(n^3 \lg m)$ time algorithm for this problem is known.

Acknowledgements

The authors thank Avrim Blum, Rao Kosaraju, Tom Leighton, Gary Miller, Mark Newman, and Baruch Schieber for stimulating discussions.

References

[ACHS88] N. Alon, S. Cosares, D. S. Hochbaum, and R. Shamir. *An Algorithm for the Detection and Construction of Monge Sequences.* Technical Report 105/88, Tel Aviv University, May 1988.

[ACY85] A. Aggarwal, J. S. Chang, and C. K. Yap. Minimum area circumscribing polygons. *The Visual Computer*, 1:112–117, 1985.

[AK88] A. Aggarwal and M. M. Klawe. Applications of generalized matrix searching to geometric algorithms. *Discrete Applied Mathematics*, 1988. To appear.

[AKM*87] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:209–233, 1987.

[AP89] A. Aggarwal and J. Park. Parallel searching in multidimensional monotone arrays. *Algorithmica*, 1989. Submitted.

[AS87] A. Aggarwal and S. Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the 3rd Annual ACM Symposium on Computational Geometry*, pages 278–290, 1987.

[BDDG85] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. J. Guibas. Finding extremal polygons. *SIAM Journal of Computing*, 14:134–147, 1985.

[CY84] J. S. Chang and C. K. Yap. A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 408–419, 1984.

[DeP87] N. A. A. DePano. *Polygon Approximation with Optimized Polygonal Enclosures: Applications and Algorithms.* PhD thesis, The Johns Hopkins University, 1987.

[EGG88] D. Eppstein, Z. Galil, and R. Giancarlo. Speeding up dynamic programming. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 488–496, 1988.

[Gil79] P. D. Gilbert. *New Results on Planar Triangulations.* Technical Report, University of Illinois, 1979.

[HL87] D. S. Hirschberg and L. L. Larmore. The least weight subsequence problem. *SIAM Journal of Computing*, 16(4):628–638, August 1987.

[Hof61] A. J. Hoffman. On simple transportation problems. In *Convexity: Proceedings of Symposia in Pure Mathematics, Vol. 7*, pages 317–327, American Mathematical Society, 1961.

[KK88] M. M. Klawe and D. J. Kleitman. *An Almost Linear Time Algorithm for Generalized Matrix Searching.* Technical Report, IBM Almaden Research Center, 1988.

[Knu73] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1973.

[LLS87] A. Lingas, C. Levcopolous, and J. Sack. Algorithms for minimum length partitions of polygons. *BIT*, 27(4):474–479, 1987.

[Mir87] A. Mirazaian. River routing in VLSI. *Journal of Computer Systems and Sciences*, 34:43–54, 1987.

[Mon81] G. Monge. Déblai et remblai. Mémoires de l'Académie des Sciences, 1781.

[SD81] A. Siegel and D. Dolev. The separation for general single-layer wiring barriers. In H. T. Kung, B. Sproull, and G. Steele, editors, *Proceedings of the CMU Conference on VLSI Systems and Computations*, pages 143–152, Computer Science Press, 1981.

[Sie88] A. Siegel. Fast optimal placement for river routing. 1988. In preparation.

[Tom80] M. Tompa. An optimal solution to a wire-routing problem. *Journal of Computer Systems and Sciences*, 127–150, 1980.

[Tou83] G. T. Toussaint. The symmetric all-farthest neighbor problem. *Comp. and Math. Applications*, 9(6):747–753, 1983.

[Wat78] M. S. Waterman. Secondary structure of single-stranded nucleic acids. In G. C. Rota, editor, *Studies in Foundations and Combinatorics: Advances in Mathematical Supplementary Studies, Vol. 1*, pages 167–212, Academic Press, New York, NY, 1978.

[Wil88] R. Wilber. The concave least weight subsequence problem revisited. *Journal of Algorithms*, 1988. To appear.

[WS86] M. S. Waterman and T. F. Smith. Rapid dynamic programming algorithms for RNA secondary structure. *Advances in Applied Mathematics*, 7:455–464, 1986.

[Yao80] F. F. Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computation*, pages 429–435, 1980.